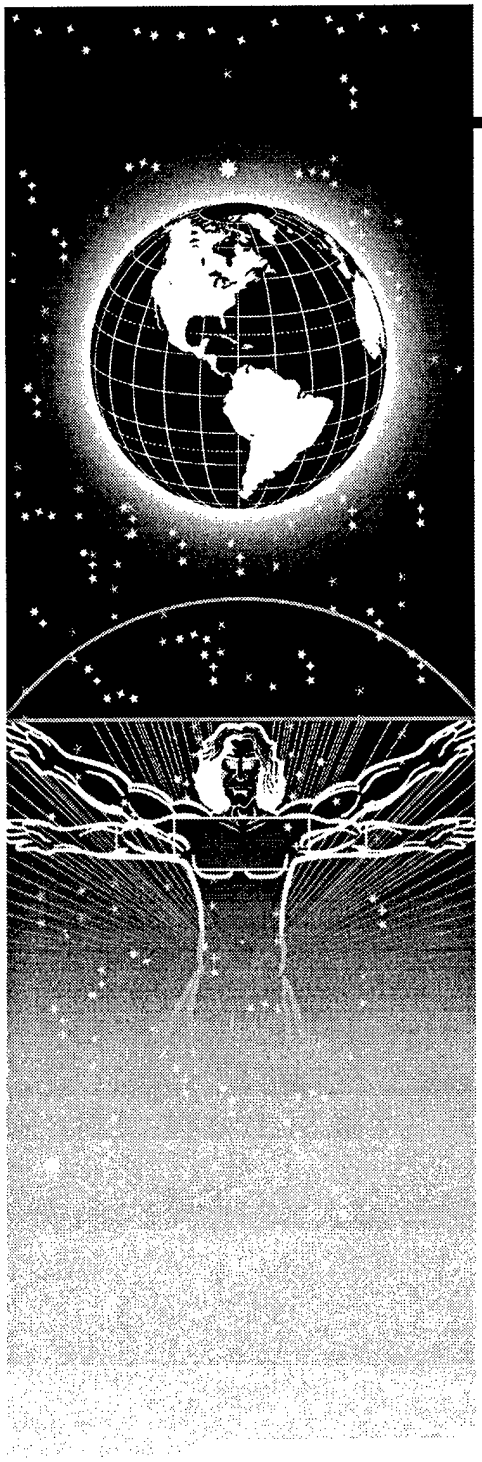
AL/CF-TR-1996-0048



UNITED STATES AIR FORCE
ARMSTRONG LABORATORY

VIDEO CHARACTER RECOGNITION SYSTEM (U)

Harry H. Heaton
Jack Houchard
Roy T. Haberlandt
Robert H. Norton

SCIENCE APPLICATIONS INTERNATIONAL CORP.
1321 RESEARCH PARK DRIVE
BEAVERCREEK OH 45432

Craig M. Arndt
Bradley D. Purvis
Stuart Turner

CREW SYSTEMS DIRECTORATE
HUMAN ENGINEERINGDIVISION
WRIGHT-PATTERSON AFB OH 45433-7022

MARCH 1996

INTERIM REPORT FOR THE PERIOD FEBRUARY 1993 TO FEBRUARY 1995

Crew Systems Directorate
Human Engineering Division
2255 H Street
Wright-Patterson AFB, OH 45433-7022

NOTICES

When US Government drawings, specifications, or other data are used for any purpose other than a definitely related Government procurement operation, the Government thereby incurs no responsibility nor any obligation whatsoever, and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

Please do not request copies of this report from the Armstrong Laboratory. Additional copies may be purchased from:

National Technical Information Service
5285 Port Royal Road
Springfield, Virginia 22161

Federal Government agencies and their contractors registered with the Defense Technical Information Center should direct requests for copies of this report to:

Defense Technical Information Center
8725 John J. Kingman Road, Suite 0944
Ft. Belvoir, Virginia 22060-6218

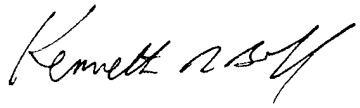
TECHNICAL REVIEW AND APPROVAL

AL/CF-TR-1996-0048

This report has been reviewed by the Office of Public Affairs (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER



KENNETH R. BOFF, Chief
Human Engineering Division
Armstrong Laboratory

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1996	INTERIM, February 1993 - February 1995

4. TITLE AND SUBTITLE

Video Character Recognition System (U)

5. FUNDING NUMBERS

C: F33615-92-D-2293
PE: 62202F
PR: 7184
TA: 10
WU: 45

6. AUTHOR(S)

* Craig M. Arndt; *Bradley D. Purvis; *Stuart Turner, Harry H. Heaton
Jack Houchard; Roy T. Haberlandt; Robert H. Norton

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Science Applications International Corporation
1321 Research Park Drive
Beavercreek OH 45432

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

*Armstrong Laboratory, Crew Systems Directorate
Human Engineering Division
Human Systems Center
Air Force Materiel Command
Wright-Patterson AFB OH 45433-7022

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

AL/CF-TR-1996-0048

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report describes the development and testing of a video character recognition system (VCRS) designed to operate in a medium to high complex, non-linear noise environment. The video character recognition system used a feed-forward, back-propagation neural network to do the recognition. The noise generation allows high recognition rates even with limited example sets, significantly increasing the utility of the system. The intended use of the VCRS is to identify character fields within video data. Specifically, it is designed to perform data extraction for flight test and photo interpretation tasks. A sophisticated user interface has been implemented so that the VCRS can be quickly customized to a new character field or new video formats.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Optical character recognition, pattern recognition, video, neural network, noise generation	260
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

This page intentionally left blank.

PREFACE

This work was performed under Air Force Workunit 71841045, "Strategic Crew Performance." It was performed by the Crew Systems Integration Branch, Human Engineering Division, Crew Systems Directorate, of Armstrong Laboratory.

The Video Character Recognition System (VCRS) was initiated as an outcome of flight test research conducted by Mr. Bradley Purvis, Lt Col Bill Marshak, Mr. Tom Green and Maj Ed Fix of Armstrong Laboratory. Collection of inflight video data from different crewstations, on missions lasting 10-12 hours, five days a week, amounted to an enormous database. Countless hours of video needed to be reviewed to obtain a few information-rich segments. A review of the postflight video data quickly revealed that the monotonous, lengthy task caused the human to make many mistakes. A better method was needed to review those high information content video segments without having to view countless hours of uninteresting video. The schema of translating video tape to video disk for random access playback was deemed a requirement. The archiving of each video disk frame to a known location on an electronic storage medium made this concept possible. The real challenge was to have the VCRS software recognize such alpha-numeric data as latitude, longitude or time stamps overlaid on the video. This proved achievable but with less than desired accuracy. The VCRS software could only attain 75% accuracy while greater than 98% was required. Further work on the VCRS will be postponed until accuracy rates can be improved.

Special recognition is accorded to the aircrew members of both the 410th Special Test Squadron of K.I. Sawyer AFB MI and the 346th Test and Evaluation Squadron of Ellsworth AFB SD.

This page intentionally left blank.

TABLE OF CONTENTS

LIST OF FIGURES

This page intentionally left blank.

LIST OF TABLES

This page intentionally left blank.

1.0 INTRODUCTION

1.1 Abstract

Character recognition in signal processing has a long and checkered history. The successes in this area have been limited to optical character recognition systems (OCR) operating in extremely low noise environments. The Video Character Recognition System (VCRS) is designed to recognize character data which has been corrupted with high levels of non-linear noise. Background occlusion noise in video presentations creates extremely complex noise patterns which render traditional algorithms useless for character recognition. The VCRS was created to help solve the problem of recognizing a limited character set in high levels of complex non-linear noise. The VCRS uses a feed-forward, back-propagation (BP) neural network to do the recognition. In order to train the network, algorithms have been developed to generate simulated noise. This noise generation during neural net training contributes to increased high recognition rates even with limited example sets, significantly increasing the utility of the system.

The intended use of the VCRS is for identifying character fields within video data. Specifically, the VCRS will perform data extraction for flight test and photo interpretation tasks. A sophisticated user interface has been implemented so that the VCRS can be quickly customized to a new character field or new video formats.

1.2 Motivation

The original requirement for the VCRS was in response to a Strategic Air Command (SAC) statement of need. SAC was attempting to utilize the B-52 as a mobile target search platform, employing the infrared (IR) and low light level television (LLLTV) steerable sensors to image targets. The Human Engineering Division of Armstrong Laboratory, Crew Systems Integration Branch (AL/CFHI) was tasked with evaluating human performance of the crews in this search task.

During the conduct of the tests, Armstrong Laboratory needed to confirm that the targets were imaged by the aircraft sensors. This was a data source required to evaluate the crew member's performance in detecting the target. Targets were known by their latitude-longitude location. In order to properly determine wether or not the targets were imaged, the precise aircraft location over the ground and the azimuth/elevation of the aircraft sensors needed to be recorded. However, no means was available for recording these parameters on the B-52. Sensor azimuth-elevation (AZ-EL) bearing was the only parameter that could be displayed on a crew member display and recorded on video tape. Thus, an airborne data collection concept evolved into a flight computer prototype.

A flight computer prototype was developed to record aircraft flight and navigation data. The flight computer provided a MIL-STD-1553 data bus to record aircraft orientation (roll, pitch, yaw). The flight computer also implemented a Global Positioning System (GPS) receiver to collect latitude-longitude (lat-long) information via an antenna protruding out the B-52 sextant port along the top of the fuselage. The sensor azimuth-elevation bearing could be recorded on video tape, along with a time stamp, and correlated post-flight data with the GPS data. The two remaining problems would be 1) the creation of data files containing sensor location and time stamps and 2) the massive amount of time required to view the video tapes. Hundreds of hours of video tape would be recorded requiring a superhuman effort to view the tape, perhaps frame-by-frame, record the AZ-EL location for each change and annotate the number alongside the proper time reference. The VCRS was conceived to alleviate this bottleneck.

The concept called for the VCRS to catalog, in real-time, the time stamp and sensor AZ-EL characters from the video tape as it was played back. The character recognizer would output ASCII files of the data read from the AZ-EL characters. This was to be correlated with GPS lat-long files. This idea proved to be more difficult in developmental practice than originally conceived.

The SAC program was prematurely terminated, but the flight computer and VCRS projects were continued by Armstrong Laboratory due to the applicability of this technology to other flight data gathering requirements and laboratory analysis applications. The flight computer was completed and qualified for flight test, however, the VCRS demonstrated a less-than-satisfactory level of performance (less than approx. 75% recognition rate). Electronic noise and variability in background scene contrast were difficult problems to overcome. After pushing the character recognition technology, and thus, VCRS performance, to a higher level, the system was baselined and further research terminated within Armstrong Laboratory because the program was growing outside of the branch charter. A suitable transition agency is being sought.

The VCRS has great applicability in the flight test community where digital flight parameters are not always available and video tape recordings of displays is the only objective data source. The VCRS would also allow analysis of previously recorded video which contains character data, such as older flight test video. With modification, the VCRS could be utilized to recognize character appearances or changes in displays, such as a target designation box, airspeed pointer, radar designation symbols, etc. Often, these types of symbols must be manually correlated with other test events during data reduction. The VCRS could save many man-hours by producing electronic files of display data.

1.3 VCRS Introduction

The various aircraft of the U.S. Air Force have evolved into complex, multi-role mission tools operated by well-trained air crew members. Most current generation aircraft have an onboard data collection capability to probe and collect avionics systems state data. Data is collected in an electronic format. The utility of analyzing electronic data for the flight test environment or the maintenance repair arena is obvious. The electronic information is not suitable for direct

analysis of air crew performance or many types of weapon deliveries. The best data for analysis is either a combination of electronic aircraft data combined with aircraft video or aircraft standalone.

Development Test and Evaluation (DT&E), Operational Test and Evaluation (OT&E), Qualification Test and evaluation (QT&E) and Tactics Development and Evaluation (TD&E) are all heavy users of video data. Other users of this medium are the Image Analysts (IAs) evaluating imagery collected from a national asset or an air breathing platform. Hundreds and sometimes thousands of hours of video tape are collected in support of the above named users. Historically, these video products have been analyzed by a qualified specialist viewing these tapes in real-time. The human is poorly suited to conduct a vigilance task such as viewing video tapes for an extended period of time and on a regular basis (Moray, 1980, 1984). Review of hundreds of hours of video tape where only a small percentage of video is of real interest leads to operator boredom and inattention. This fact led AL/CFHI to conceptualize an electronic system to perform the video search task automatically.

The concept, as envisioned by AL/CFHI, requires input of the video source into a character recognition device that is capable of reading and understanding the characters on the video tape. The information is then stored in a random access device with the location of the random access device identified and cross correlated for retrieval. Therefore, the operator of such a system would be able to input a video source tape to the automated system, the system would read the data in real-time from the source tape and correlate that frame of data containing characters to a random access site. The human operator would be required to manage and supervise the automated system, such as loading a new tape once the old one is archived or determining if the random access storage device has sufficient space for follow-on data collection. The strengths of both an automated system and the human operator together optimize task performance. Once all the video tapes are translated to a random access storage device, operator manipulation is possible. For example, most flight tests record time and/or lat-long information. Thus, review of a particular flight test segment or area of interest is then possible by recalling either the time and/or lat-long. A system display with a menu driven screen would provide an acceptable operator interface to selectively retrieve information from the stored mission video.

This page intentionally left blank.

2.0 DESIGN REQUIREMENTS

2.1 Background

2.1.1 Optical Character Reader

OCR is used to transform information from a visual domain into binary code useful for machine processing. Commercially, OCR is used to scan printed pages for conversion to ASCII text, thus replacing the manual task of keyboarding hardcopy documents. The operation is performed by scanning the hardcopy to produce a digitized (either gray scale or binary) image, which is then processed to recognize individual characters, symbols, and white space elements.

While posing a challenging technical problem, OCR technology has only recently proven useful on a broad scale, with error rates dropping to a level comparable to a skilled typist. Historically, OCRs have been sensitive to the quality of the input image, often requiring "clean" original type from a high quality printer. In a review by Byte magazine conducted in 1991, recorded optical character recognition rates were below 60%. More robust performance is available in current commercial offerings, with tested recognition rates ranging between 99.3% for daisy wheel text to 78.1% for faxed pages (Byte, October 1994). In addition to classic image processing and rule-based recognition methods, a top rated commercial system, OmniPage Professions, incorporates a neural model in their OCR engine to improve recognition rates. Sabourne and Mitiche (1992) report OCR recognition rates with an entirely neural-network based system of 96.7% for characters selected from 200 fonts and 50 typical documents.

The problem domain for the VCRS, although related, differs significantly from the commercial OCR domain. Commercial OCR systems can expect relatively high quality, high contrast input, but have a complete recognition problem that spans the multiple fonts, characters and symbols encountered in modern word-processor documents. In support of this problem, standard training sets exist consisting of literally thousands of exemplar patterns.

In contrast, the VCRS task is to recognize but a single font and a limited number of symbols of vastly inferior image quality compared to the input required for commercial OCRs. Each video character consists of a low number of pixels. The characters are often ill-formed due to the distortion inherent in the video source, and rather than being presented against a uniform background, the characters overlay a complex, highly patterned background formed by sensor imagery.

2.1.2 Neural Networks

Artificial neural networks are computational systems that are loosely based on our understanding of biological neural systems. In biological systems, simple computational units (neurons) are densely interconnected by synaptic junctions with other neurons. This being said, it should be

noted that exceptions are the rule in biological systems. Biological neurons and their interconnections form complex families which differ both morphologically and functionally. Artificial neural networks are by necessity, highly simplified, idealized approximations of their biological counterparts.

Artificial neural networks can be applied to two overall classes of problems: data clustering and pattern recognition. Network paradigms associated with data clustering use unsupervised learning mechanisms to group related data present in input vectors into a number of output classes. Classic examples of such networks include the Kohonen feature map and the Adaptive Resonance Theory (ART) networks (Wasserman, 1989). In the second overall area, pattern recognition, supervised learning mechanisms are used that are variations on the classic "Perceptron" neural network. The most widespread paradigm of this type is the back-propagation network.

"Back-propagation" is the term applied to a closely related family of networks which are configured as feed-forward networks trained through a back-propagation of output errors. All use a gradient-descent form of learning, with the most common example being the generalized delta rule (Rumelhart and McClelland, 1986). Figure 2-1 depicts a simplified organization of a typical back-propagation network. The circles represent the "processing units" (neurons), while the interconnecting lines represent "connection weights" (synapses).

All processing units perform the same task: sum the inputs provided to it and provide an output that is a function of the input sum (the function used is typically a sigmoid logistic non-linearity -- the generalized delta rule requires a continuous differentiable non-linearity). The output from each processing element is fed forward to the next tier of processing elements via the interconnections. Each interconnection has a corresponding "weight" which represents a multiplier (either positive or negative) for the output from the first layer. This process continues for an arbitrary number of layers of processing elements until the last layer is reached. The output values of these processing elements represent the output for the overall net.

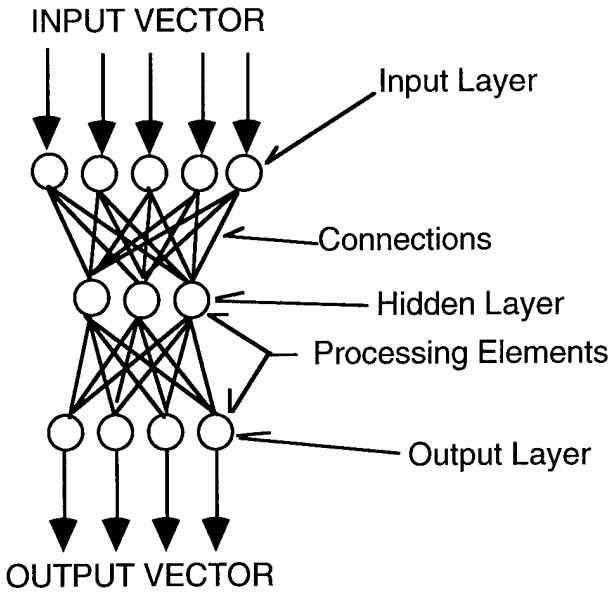


Figure 2-1. Configuration & Nomenclature for Typical Back-Propagation Network

In its initial, untrained state, the neural net's interconnection weight values are randomized. The network is trained by presenting both an input pattern and a desired output pattern (the exemplar set) to the network. The pattern is processed (summation and output) by the nodes of the input layer. The output of this step is modified by the connection weights and fed-forward to the hidden layer, where the inputs are again processed. This process is repeated on until object (or character) classification occurs at the output layer, where the resulting output pattern is compared to the desired output pattern. The differences between the actual and the desired pattern are used to adjust the connection weight values back through the network (back-propagation of errors). The process is then repeated with another exemplar pattern.

Through repeated training in this manner, the interconnection weights of the network are incrementally adjusted until the desired output pattern is produced (within an appropriate tolerance) for each exemplar pattern that is presented. In this manner, the information required to produce a desired pattern is present in the interconnection weights. All processing elements, or nodes, simply perform the same sum-and-output operation on the values presented at their input connections.

The three-layer network depicted in Figure 2-1 has the benefit of allowing the development of an internal representation in a hidden layer, allowing the re-mapping of an arbitrary set of inputs to an arbitrary set of outputs, provided there are sufficient hidden layer processing elements. While it can be shown that a three-layer network will solve such re-mapping problems, there is no guarantee that this architecture is the most efficient solution. Networks containing multiple hidden layers are not uncommon.

2.2 Video Character Recognition Task

This section discusses VCRS system requirements, the hardware component configuration used to support system development and the VCRS modes of operation.

2.2.1 Hardware Requirements / Configuration

The critical VCRS hardware components, their function, and the related performance characteristics are described below.

2.2.1.1 Modes of Operation: The hardware configuration selected was based upon operational mode requirements. The VCRS modes included the following:

a. Video record

b. Exemplar set extraction

c. Neural net training

d. Character recognition performance evaluation

2.2.1.1.1 Video Record Mode: The video record mode entailed a video tape-to-optical disc record process. The optical disc media was selected as the format to support video playback for three reasons: 1) a stable video sync during video playback, 2) a computer control interface and 3) a random video frame select capability.

Video tape record/playback (VTR) technology cannot provide a reliable video frame sync without time base correction (TBC). Thus, the record process employed a TBC/synchronizer component to provide a relatively stable sync signal in order to minimize video signal jitter. See page 3 of "SCB-100N Video Tutorial" (The Grass Valley Group, Inc., 1987) for a discussion of time base correction of video. A measurement of jitter indicated a horizontal line shift of approximately 10 nanoseconds. A VTR jog shuttle function supported the selective frame process of recording Super-VHS (S-VHS) video onto the analog optical media.

The hardware components critical to the video recording process included the following.

a. Digital Processing Systems TBC/Synchronizer, Model 265

b. Panasonic S-VHS Video Recorder/Player, Model AG-5600

c. Panasonic Optical Disc Recorder/Player, Model TQ-3031F

Figure 2-2, is a photograph of the component hardware. Figure 2-3 depicts the component configurations and their interface definition. The Panasonic optical recorder provided a S-VHS resolution greater than 400 lines. Thus, the loss of image content during recording was minimized.

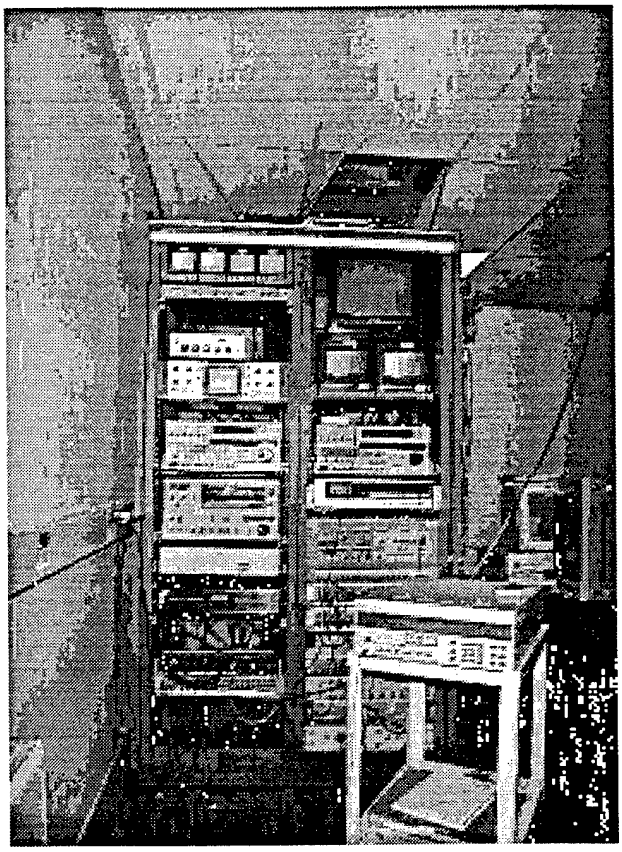


Figure 2-2. Video Recording Equipment.

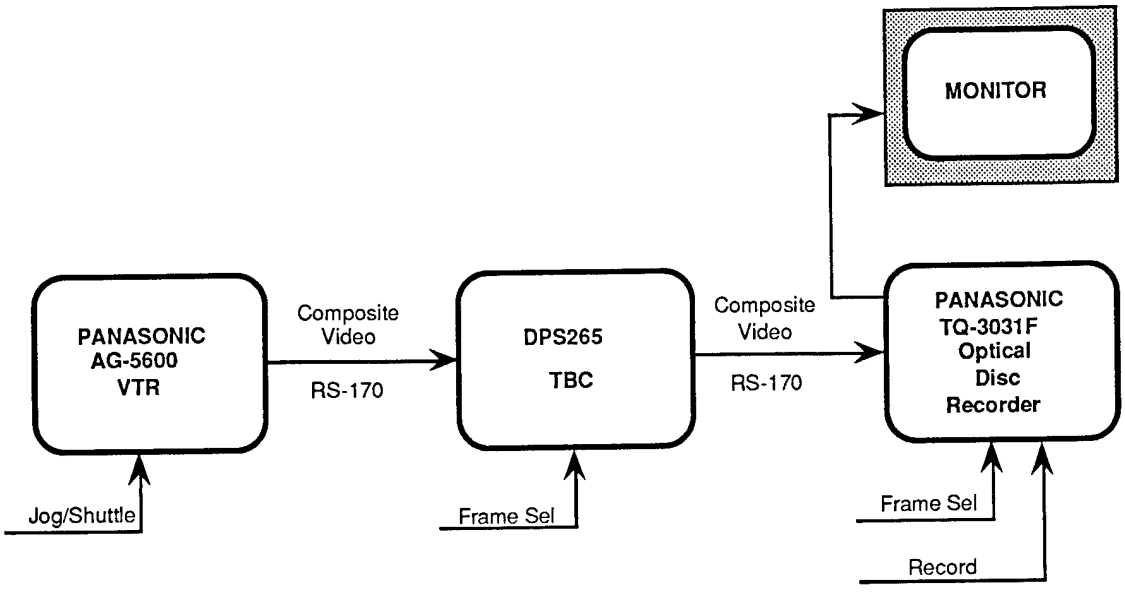


Figure 2-3. Video Record Mode

The computer control interface of the optical disc recorder/player could have supported the automated playback of video under computer software control. This implementation was planned to support a character recognition-evaluation mode requirement, however, data collection performance in a frame-by-frame fashion precluded this implementation. The third reason for using the optical disc player was for ease of use during the process of building exemplar data sets. The capability to randomly select characters between video frames provided a time efficient means of exemplar development.

2.2.1.1.2 Exemplar Set Extraction: The Exemplar set extraction mode entailed the use of a personal computer (PC) -based video capture card to digitize video frames played back from the optical disc player. Figure 2-4, Exemplar Mode, is a functional block diagram. Process-critical requirements included: 1) minimum 8-bit resolution, 2) computer controlled optical disc playback, 3) RS-170A video format and 4) high performance video imagery processing function. Thus, a critical hardware component, in addition to the optical disc player, was the video frame grabber. The frame grabber selected was a Modular Frame Grabber (MFG) with Color Acquisition Module (AM-CLR) and was developed by Imaging Technology.

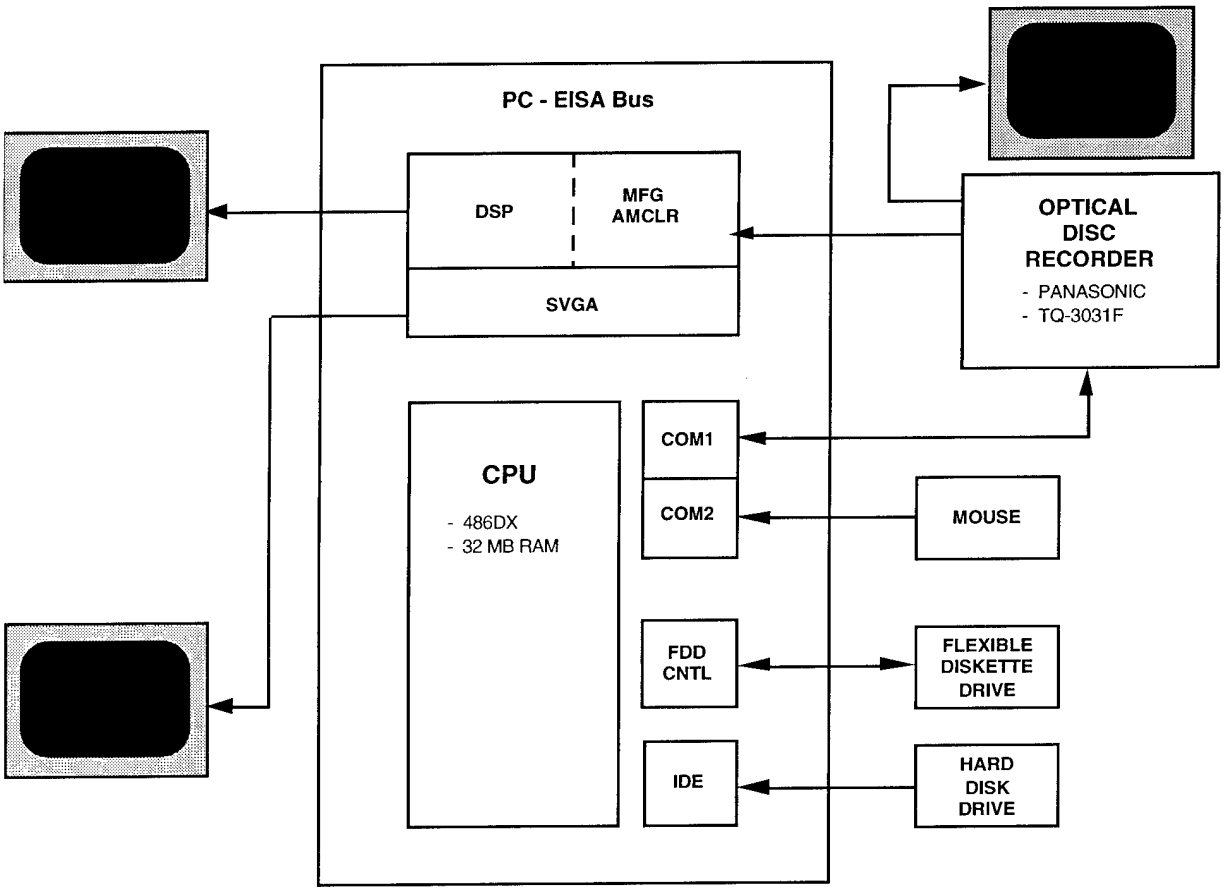


Figure 2-4. VCRS Exemplar Mode

The video sources selected for the VCRS task contained FLIR imagery with text overlay. The MFG frame grabber supported 8-bit resolution of each video pixel to minimize loss of gray scale content. In addition, the MFG frame grabber provided a high performance, video imagery processing capability. This capability was provided with an integrated, onboard, DSP component, the TMS 34020 Graphics System Processor developed by Texas Instruments. The programmable DSP function off-loaded the image processing task from the host central processing unit (CPU). Thus, VCRS screen update performance was maximized in comparison to the host CPU being burdened with both image processing and screen update transferral via the computer system bus.

The PC platform with the MFG frame grabber function operated in a dual monitor mode as a matter of convenience. One VGA monitor was used as a DOS interface, while a second high resolution monitor was dedicated to the VCRS graphical user interface. Figure 2-5, below, shows the dual monitor mode arrangement employed in the laboratory. See Appendix A, VCRS User Manual, for VCRS screen formats. Digitized video was displayed in a window, 512 x 512 picture elements (pixels), within the VCRS interactive screen. The random frame select function provided by the optical disc playback unit supported ease of use during the development of an exemplar set from various frames.

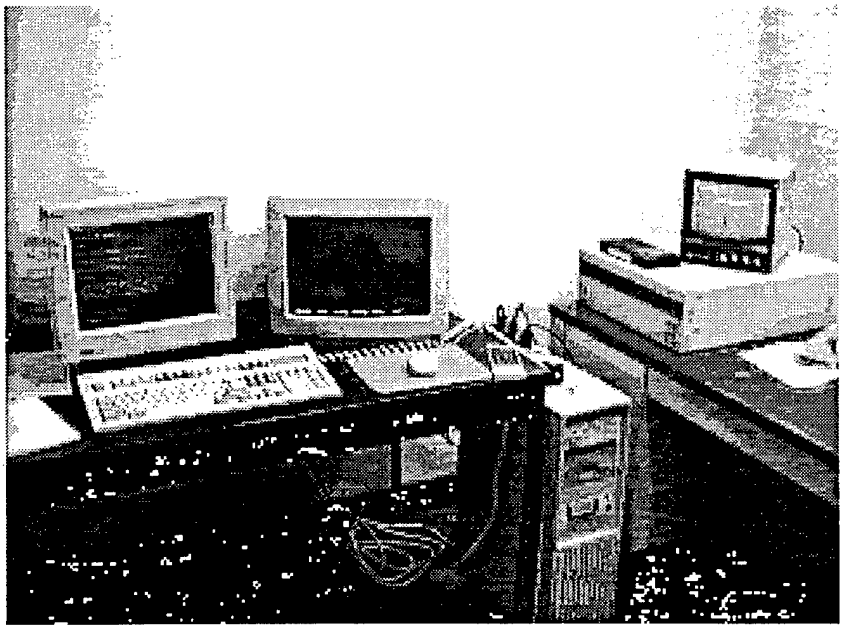


Figure 2-5. VCRS Dual Monitor Mode

2.2.1.1.3 Neural Net Training: The training mode of a neural net is an iterative process which places a significant demand on computer processing time. Today's fourth generation general purpose microprocessor provided a cost effective solution to meet training process

requirements. An IBM compatible PC, with an Intel-based 486DX CPU operating at a 50 MHz was selected as the VCRS platform. The PC hosted the Microsoft disk operating system (DOS), version 5.0. The PC configuration included the following peripherals: 213 megabyte (MB) hard disk drive, a floppy disk drive, a Microsoft mouse and a 101-key keyboard.

2.2.1.1.4 Character Recognition-Evaluation Mode. Preliminary requirements for the character recognition-evaluation mode dictated that the video playback unit support a software driven frame select function. This function was intended to support an automated frame select process under VCRS computer control. This interface was prototyped but not applied. The automated video frame stepper was not used since the collection of neural net performance data was done in a single frame by frame fashion.

2.3 Theoretical Development

In traditional neural network applications, a large amount of data is required in order to define the entire "destination region" for the network. There are, however, a large number of problems for which it is difficult, inconvenient, or dangerous to gather large numbers of data points in order to use a traditional network training algorithm. In order to use smaller example sets for training, several different approaches have been proposed.

The most popular approach has been to create a mathematical model of the system the neural network is studying and then generate training sets based on perturbations of these models. This approach works well for highly complex systems in which there are numerous interactions which generate the data sets.

In the case of systems with non-complex data but highly complex noise, a new approach is proposed. Highly complex noise patterns can be simulated by a combination of random and high order, structured, non-linear noise. In the case of two-dimensional character data, a "blob" generator is employed for non-linear structured noise.

3.0 DESCRIPTION OF IMAGE DATA

3.1 Imaging Sensor Video

The source imagery for the exemplar and test set was obtained from cockpit video tapes of a Forward Looking Infra-Red (FLIR) video originally obtained from a prototype "Falcon-Eye" system. This system provides a motion-stabilized image, generated from a linear Infra-red (IR) detector array. In the F-16 cockpit, sensor video as displayed on the MFD is scan-converted and captured on video tape (either 3/4" cartridge or 8mm formats, depending upon the individual aircraft). Video data from several other sensors as well as simulators was available. However, no attempt was made to train a network to recognize fonts from different video sources. The FLIR source offered a large number of video frames with a great deal of variation in image quality.

3.1.1 Media Stabilization

Video tape imagery characteristically has line-by-line and frame-to-frame timing variations due to stretch and distortion in the recording media. While not noticeable on a conventional monitor, these variations preclude directly grabbing frames (referenced against a microprocessor's time base) from the original video tape media. To remedy this, the video tape data was re-recorded onto a Panasonic optical disk after being stabilized with a digital time-base corrector. The resulting series of video disk images had sufficiently low timing jitter to allow frame grabbing, and were used as the source of both exemplar and test data in subsequent trails.

3.1.2 Character Extraction

Characters were extracted from each video source by defining specific windows where the characters occurred as depicted in Figure 3-1 (the user interface to define and configure such windows using the VCRS software is described elsewhere in this report). Although the VCRS software allowed the re-sizing of large characters to reduce both storage requirements and input vector size, this function was not used. Instead, frame boundaries were selected which were close to (within 1 or 2 pixels) the bounds of characters in the display. This resulted in an extracted image of 19 pixels wide by 31 pixels high by 8 bits deep for each character. None of the extracted character windows were re-sized to avoid the loss of information that occurs in using nearest-neighbor interpolation. For large characters that are reduced in size, this can result in the elimination of high frequency, horizontal or vertical components of the image.

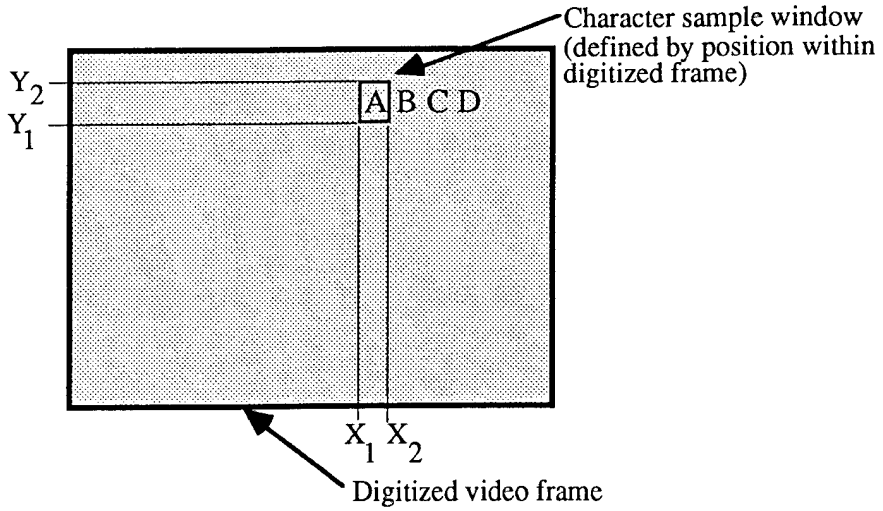


Figure 3-1. Definition of a Sample Window Within a Frame

Figure 3-2 depicts a sample of typical characters from this video source, along with samples of typical modes of distortion. Several general sources of distortion/corruption exist in all samples from this video source.

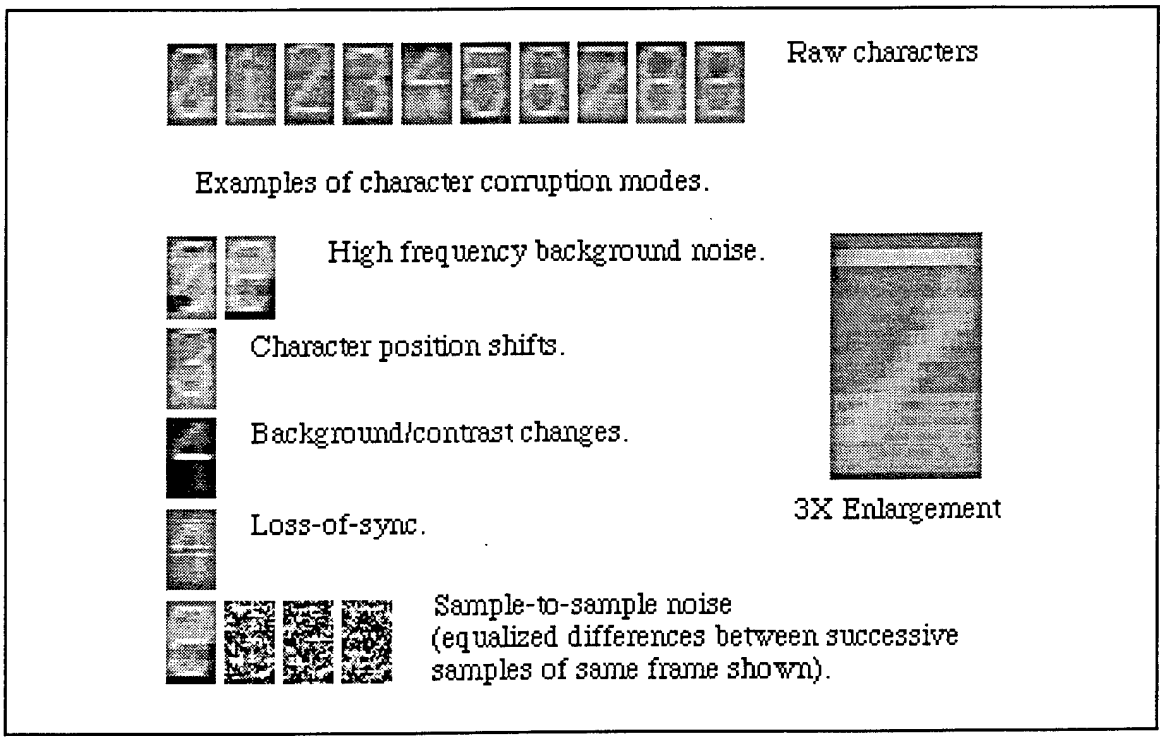


Figure 3-2. Sample FLIR Source Data

3.2 Image Characteristics

Note that in all characters, vertical lines in the characters are less intense than horizontal lines. This is believed to be a function of the voltage swing time of the video amplifier in the original signal source (although it could have been introduced at any one of several stages in the video recording and transfer process). Figure 3-3 depicts the output voltage achieved in an amplifier of limited voltage slew rate in response to a step change. In attempting to generate a single white pixel on a black field, our example amplifier (20 milli-volt/second slew with pixel size representative of an 875 line, 60 Hz display) fails to reach saturation. However, if several white pixels are contiguous on a line, the amplifier does reach saturation. Close examination of the components in the enlarged "7" in Figure 3-2 reveals a clear ramp-up/ramp-down in intensity in the top, horizontal component, consistent with amplifier response limitations. The down-ramp at the right side of the character actually extends beyond the vertical component of the character, indicating slow roll-off in the amplifier.

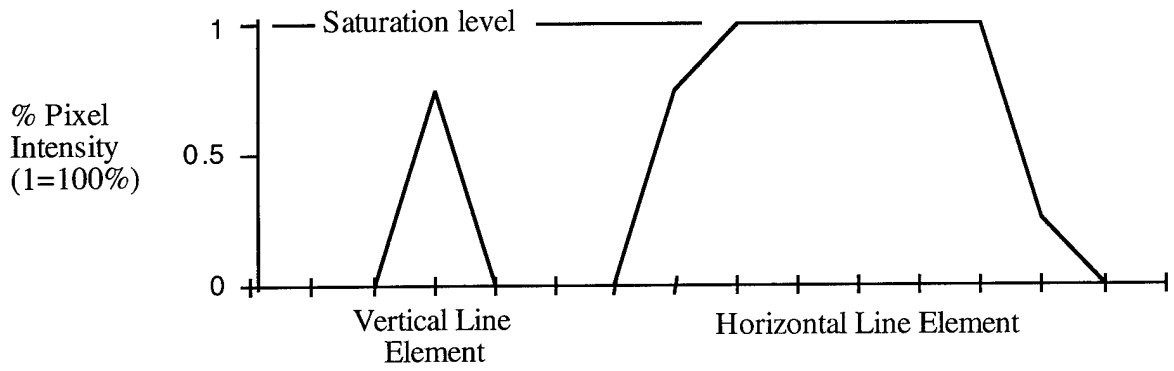


Figure 3-3. Example of Video Amplifier Response

Note also that the short, purely vertical component is barely separated from the background (actual difference in pixel value for this sample is 23 out of a possible 256 gray levels). For vertical components in general, this magnitude difference is maintained. For example, as seen in the sample character "4" in Figure 3-2, the change in pixel values between the dark background and the brightest portion of the vertical component is 28 gray levels.

This suppression of vertical components complicates the recognition of characters in this set. Many of the characters are structurally similar. The characters "6," "8," and "9" differ primarily by a single vertical component. Several examples exist in the video where the experimenter cannot determine, even on close inspection, whether a specific character is an "8" or a "9."

Another general feature of this signal source is the horizontal banding present throughout the image. This is due to the nature of the sensor, which optically sweeps an image field over a linear detector array. The banding corresponds to individual detectors in the array.

Several other types of distortion illustrated in Figure 3-2 occur on a character-by-character to frame-by-frame basis. Background noise occurs whenever the character field overlays background features in the sensor field-of-view. Character position shifts of one or two pixels are common. Extreme contrast changes are possible, either due to background scene content or pilot control inputs (i.e., selecting white/hot or black/hot on the FLIR display). Also in the time base correction process, extreme variations in timing on the video tape are frozen into the double-image depicted in Figure 3-2. Finally, sample-to-sample noise is present, representing both random and periodic system noise. The images shown are equalized to show the distribution of noise in successive samples. The maximum magnitude of the noise is limited to about 32 bits, with a roughly Gaussian distribution about zero.

Because of the complexity of the image, some common approaches in image recognition cannot be applied. One example is dynamic sizing of the sample window based on the detection of edges. This is often applied in handwriting recognition applications where the size or position of entry data varies. However, in the VCRS application, the character video in the data field overlays a complex video image. This renders edge detection approaches to find the limits of the video character unworkable: the edges found may as easily belong to the background as to the character.

4.0 HISTORY OF SOFTWARE DEVELOPMENT

4.1 Software Development

The initial program objectives were to develop a tool for extracting and recognizing data annotated on video. A need arose during a flight test program where certain data was required for post test analysis, but could only be captured as annotated data on a video recording of sensor imagery. Early requirements focused on this capability, including 1) the ability to grab and digitize video of multiple formats, primarily 875 line RS-343 and both super and normal VHS, 2) extract matrices of pixel luminance values from the digitized frames, 3) recognize the characters contained in the matrices, and 4) record these in a well defined format for later processing. While there were no fixed requirements for the speed of the process, it was hoped that the process could be made to run in real time for reasonable extraction requirements (30 Hz).

Budget constraints forced the selection of commercial off the shelf hardware, as opposed to custom designed hardware, to satisfy these requirements. The requirements dictated the use of high-end (in terms of cost and performance for 1990 technology) hardware, creating numerous problems during software development due to the limited availability of device driver software which met the hardware performance needs.

Three ground rules were established for the software development effort: 1) the annotated data appeared in every frame; 2) the data was in a fixed position and format on each video frame; and 3) font size was fixed (i.e., not proportional). These ground rules eliminated the need to search a given image for the annotated data, dramatically speeding up the process of acquiring and recognizing the character data. Armstrong Laboratory had developed a neural network-based character recognition and training algorithm which was to be used for the character recognition requirements.

The initial software development effort concentrated on developing four basic capabilities.

1) The user needed the ability to identify the location and characteristics of the annotated data to be parsed and recognized. This included the identification of fields of characters, e.g., the latitude field or the longitude field, the number of characters in each field and the size of the characters. This evolved into the current ÒField Editor,Ó which presents still frame video to the user and allows the user to identify the position and characteristics of the characters in each field. During development testing with real video from multiple sources, the distance between characters in a particular field varied by one or two pixels due to timing of the digitization process. Hence, it was necessary to allow the user to reposition individual characters in a field, to get the actual character centered in the pixel matrix which would be frame-grabbed. In one of the Heads-Up

Display (HUD) videos, it was also found that different fields could be in different fonts; consequently, software was added to define character sizing information.

2) Early in the development process, it was discovered that a single neural network could not be used across all of the sources of video, and font types. Consequently, a capability to create custom neural networks for each video source was constructed. This required the ability to create an exemplar set for neural network training which evolved into the current Òexemplar editor.Ó Initially, the exemplar editor was used to extract a given pixel matrix and identify the specific character the matrix contained. This had to be done for every character in the set to be recognized, but it only had to be done once to characterize a given video source. However, as described in (3) below, it became necessary to edit each characterÕs pixel matrix to create a ÒperfectÓ high contrast character, which could then have various types of corruption superimposed upon it.

3) Initially, the neural network trainer was created to ÒwrapÓ the Armstrong Laboratory training algorithm for this application. The AL algorithm used additive random noise as the primary technique for improving neural net robustness and generality. This technique was sufficient for very high contrast characters (i.e., white on black), but had very poor recognition rates when used on actual imagery, where the background luminance varies greatly. When HUD imagery was examined and tested, the background could contain luminance gradients, partial lines of demarcation, such as a building or shoreline, etc. and/or distinct or indistinct blobs, such as sagebrush or rocks. Also discovered, as an artifact of the original frame grabber, was a total character position shift in the pixel matrix by up to 2 pixels, thus destroying recognition performance. Consequently, the training algorithm was modified to incorporate various types of "corruption." The current algorithm can train on various foreground (character) luminance values, background luminance values, ÒblobÓ corruption (randomly drawn concave and convex polygons of from 3 to 8 vertices), position shifting, as well as the original additive noise. During this evolution of training algorithms, it was necessary to edit exemplar characters to provide a high contrast, ÒperfectÓ character, on which the corruption techniques could be superimposed. The trainer was designed so that the user was periodically shown the actual results of training and net performance, thus providing visual feedback of training progress.

4) Finally, once the network was built, some mechanism for actually performing the character data extraction was necessary. An initial capability was built which provided reliable runtime and recognition rate performance estimates. The intent was to incorporate these same capabilities into a stand-alone program which would run as fast as was possible based on the computing technology available.

During the development process, it was also discovered that the image quality resulting from the digitization process for a given tape could vary greatly. This was a problem with the supporting software for the frame grabber. The only way to circumvent this particular problem and get

consistent image quality was to create a frame editor, i.e., an editor which allowed the user to control and save the setup data for the frame grabber board, including the timing data, maximum and minimum luminance, and horizontal and vertical standoffs for the video source.

These capabilities were initially developed into separate, stand-alone applications. As the software evolved (due to better understanding of application requirements and the vagaries of reality), a common set of user interface requirements emerged, and it became more practical from a maintenance standpoint to integrate the applications. The integrated application was designed to be a windowing application without Windows, operating on a message-driven state machine.

4.2 High Level Structure

The following figures and process specifications depict the high-level structure of the VCRS software, corresponding to the source code listings found in Appendix C. The figures use the Data Flow Diagram (DFD) with real-time extensions as described by Hatley-Pirbhai (1988). DFDs use a series of graphic elements to depict various software structural elements as described below.

- Source/Sink: Rectangular boxes represent data sources or sinks. Sources and sinks are external to the software system.

- Process: Whether depicted as solid circles or ovals in the diagrams, the meaning is identical. Bubbles define software processes where incoming data is modified to form a new, output data flow.

- Data Flow: Depicted as a solid arrow, these lines depict the name and direction of data exchange between other elements in the diagrams. The data flows in these diagrams may represent either a single data element, also called a primitive, or a collection of related data elements, called a composite data flow. There is no graphic distinction between a primitive and composite data flow. Each data flow is named, and can be traced from one level of the diagram to the next.

- Control Flow: Depicted as a dashed arrow, this represents discrete events required to activate a process bubble.

- Data Store: These structures are depicted by parallel lines enclosing the names of data stores that are internal to the software process. Stores represent data structures defined and modified within the software process.

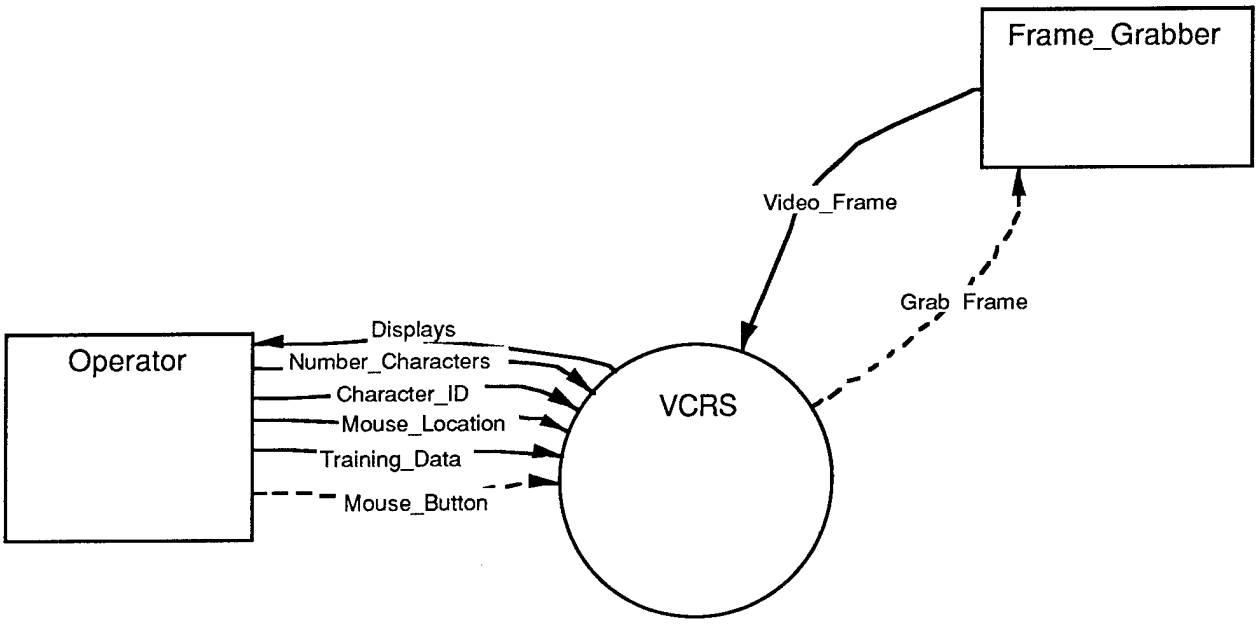


Figure 4-1. VCRS Context Diagram

The first diagram, Figure 4-1, is the context diagram, which provides a summary view of the overall system. In the context diagram, the central bubble encompasses the entire software system, while all external elements are sources and sinks. In this report, the decomposition of software processes is kept to a high level, and is presented solely as an aid to understanding the source code. The context diagram defines the limits of the system in terms of inputs and outputs. Note that all inputs and outputs at this level are limited to the operator and the video frame grabber.

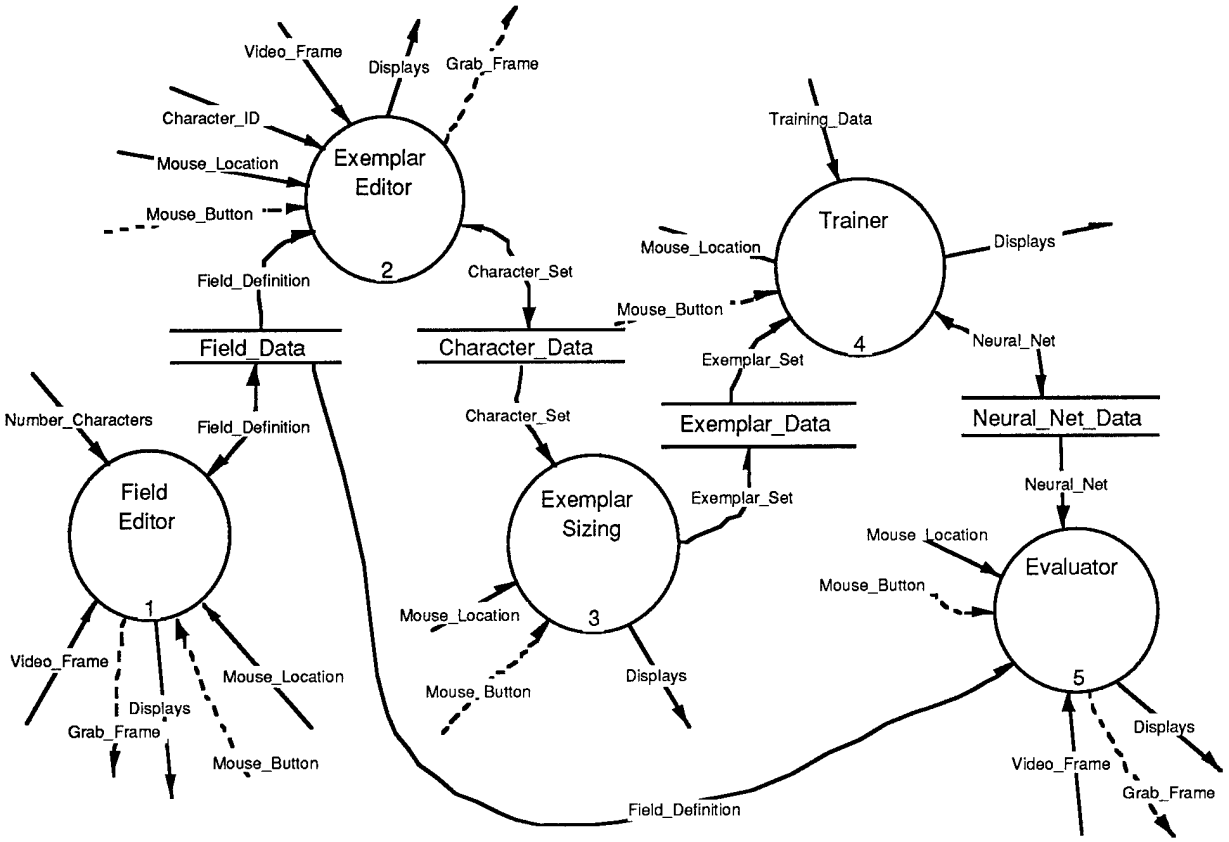


Figure 4-2. VCRS Level 1

As shown in Figure 4-2 above, the level 1 DFD for the system is located inside the central bubble of the context diagram. Defined here are the five major processes that make up the VCRS software system. Each of the process bubbles are numbered. Complex processes are further exploded in the following figures, while relatively simple processes refer directly to a process specification. For example, the Exemplar Editor processes are not further exploded since they represent a relatively simple pixel-editing process. A process specification is a high-level text description of the software process that takes place within a bubble.

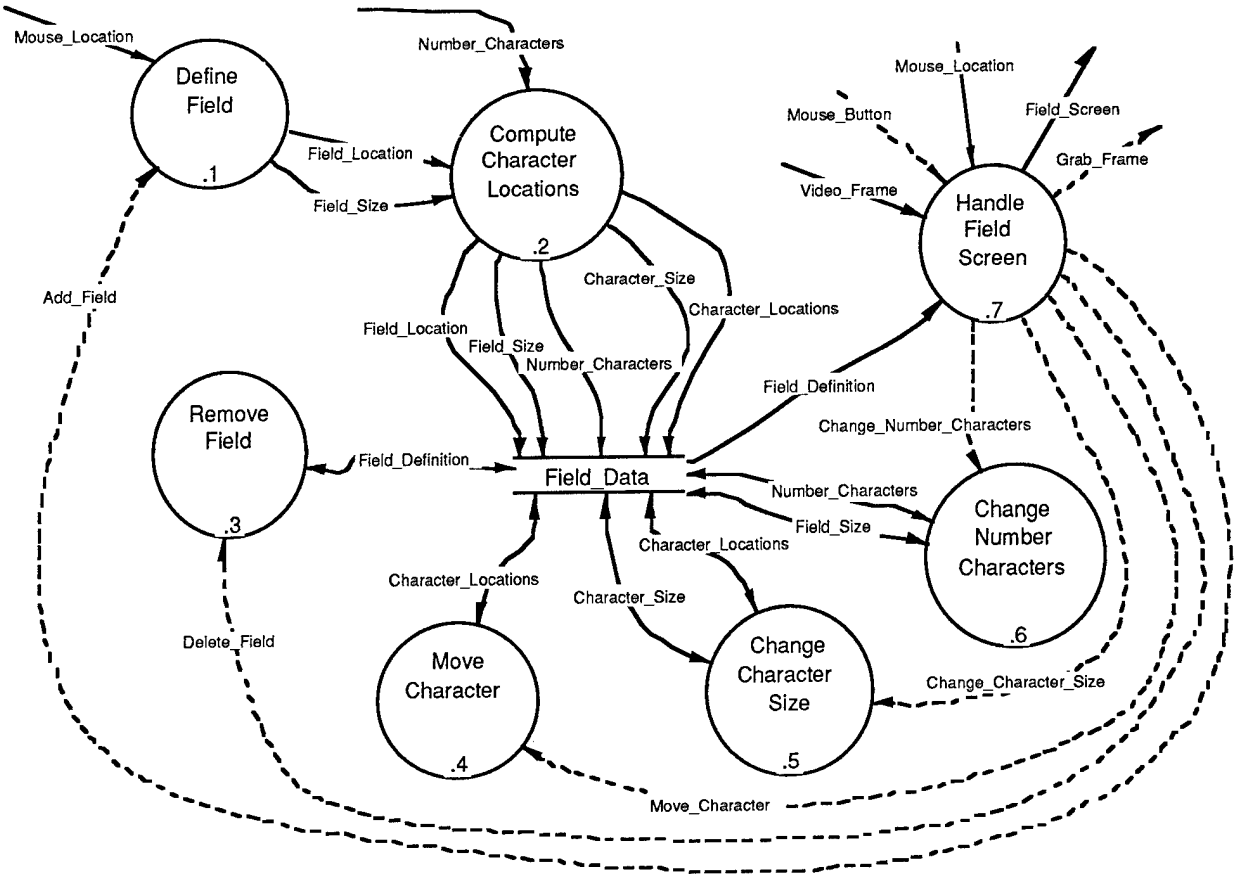


Figure 4-3. VCRS Level 2, Field Editor Processes

Figure 4-3 shows the level 2 DFD depicting the sub-processes contained within the field-editor process. These processes define the graphic user interface operations used to define the pixel fields extracted from a grabbed frame for each exemplar character.

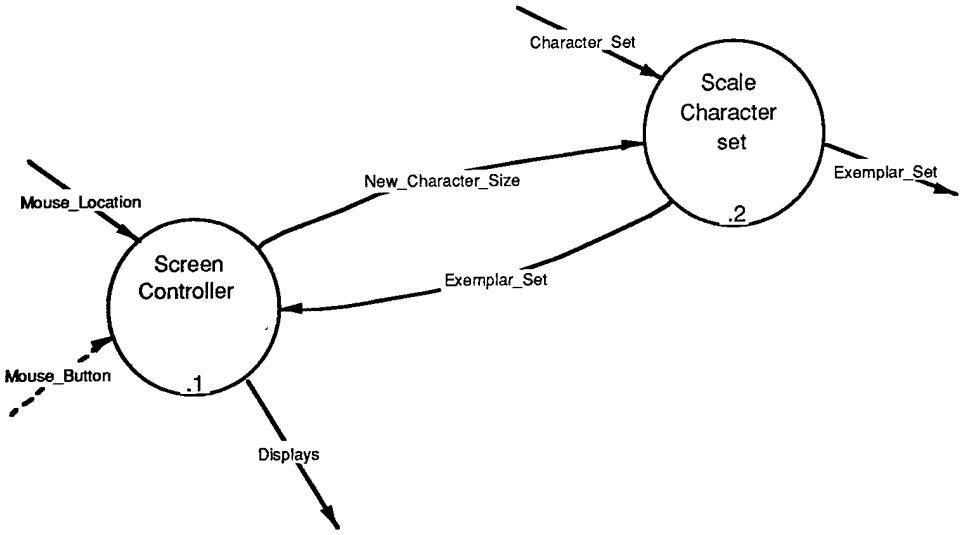


Figure 4-4. VCRS Level 3 Exemplar Sizing Processes

The Exemplar sizing process, depicted in Figure 4-4, defines the methods by which defined pixel fields obtained from the frame grabber are re-sized to form exemplar sets of a uniform size. Exemplar sizing was 1 to 1 with the grabbed field for the evaluation of the network presented in this report.

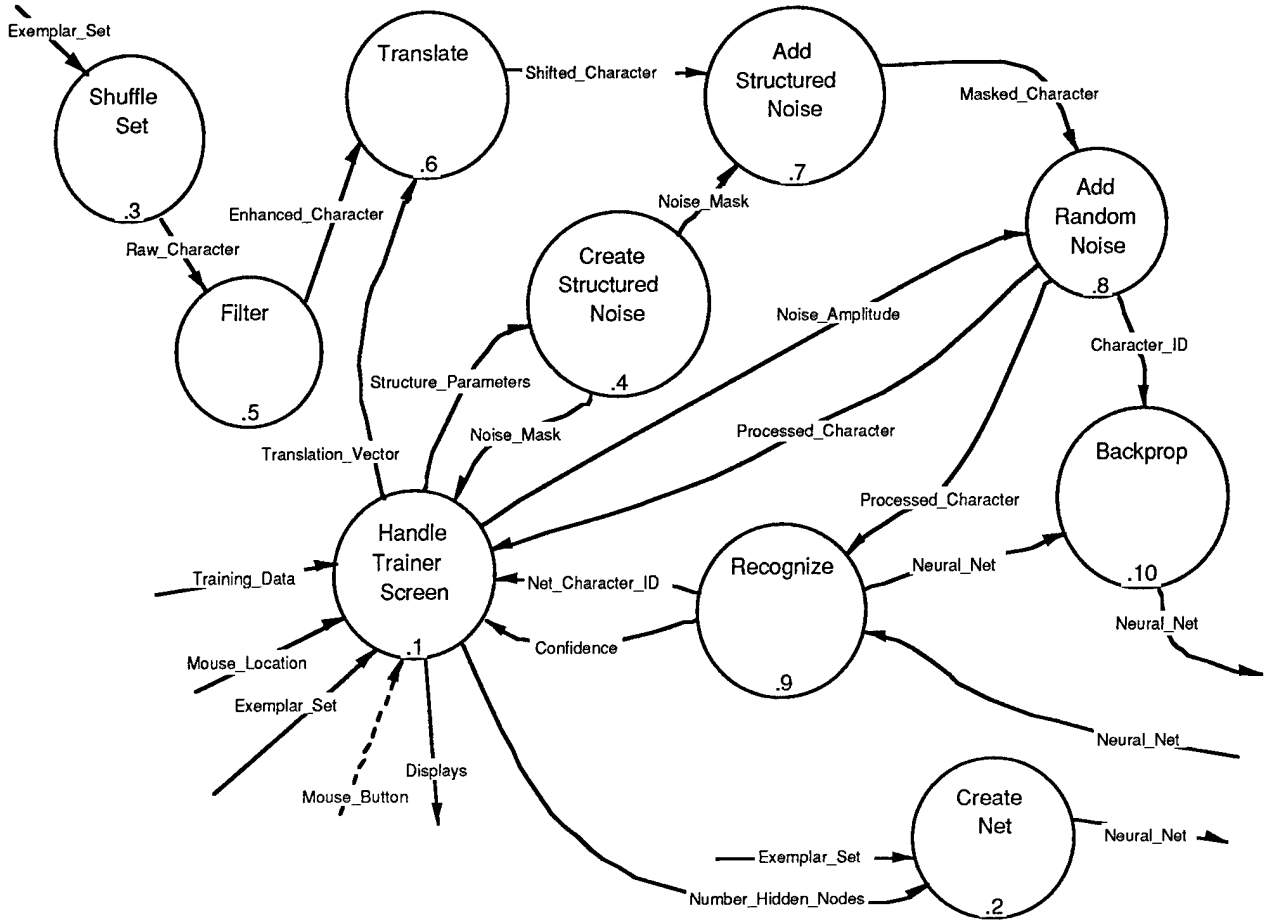


Figure 4-5. VCRS Level 3, Trainer Processes

The trainer processes, shown in Figure 4-5 above, include the basic back-propagation processes used to train the neural network as well as all of the random corruption algorithms applied during training to artificially expand the exemplar set..

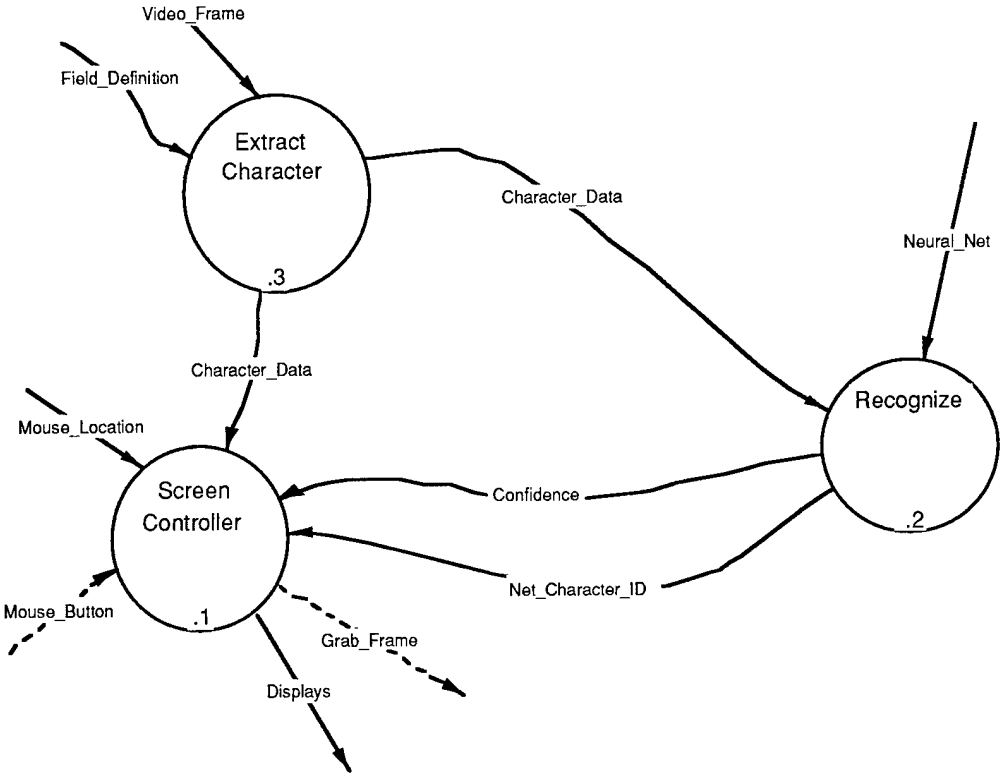


Figure 4-6. VCRS Level 3, Evaluator Processes

The evaluator process Figure 4-6 above, defines the sub-processes used to perform character recognition on grabbed fields after training is complete.

This page intentionally left blank.

5.0 SYSTEMS DEVELOPMENT

5.1 Network Paradigm

A feed-forward network with back-propagation of errors (BP) neural network paradigm was used for this study. This paradigm was selected for a) being relatively mature, with well understood behavior, and b) itÕs ability to handle gray-scale vectors while taking advantage of existing BP code developed by AL/CFHI. The reuse code was modified to implement variations to the standard BP training to accommodate variations in the image data.

5.1.1 Exemplar Set Perturbations

The user interface to the training portion of the network allowed random noise, blob noise, and pixel shifts to be implemented to a user-defined degree during training. Each of these is described further below.

5.1.1.1 Random Noise

It is well known that the addition of some degree of random noise to the input vector improves the training speed and pattern recognition performance in BP nets. During training, a BP network uses gradient descent learning to adjust interconnection weights from an initial random state. This form of learning is readily "trapped" in a local minimum, well short of a generalized solution. The addition of a percentage of random noise to the input vector reduces the effects of local minima, improving convergence behavior by reducing the differences between the discrete samples in the exemplar set. The addition of random noise on a pixel-by-pixel basis to the exemplar set during training was implemented as a user-defined value between 0 and 255.

5.1.1.2 Blob Noise

During initial development of the VCRS software, development trials revealed that while random noise improved convergence on the exemplar set, it had less impact on the recognition accuracy for images outside the exemplar. Examination of the FLIR images found that the background video often contained highly structured, high frequency data. For example, in otherwise uniform desert terrain, bright "blobs" of desert plants would move through the character fields as the aircraft flew. Since an exemplar set depicting all such variations would be prohibitively large, an approach was developed to degrade the exemplar data with random "blobs."

Public domain code was identified which could fill a concave polygon defined by a random number of points on a plane. This code was adapted to generate a complex polygon with the bounds of an exemplar image, filled with a single pixel intensity set randomly within a user-defined range. The resulting polygon was added to the exemplar image if the fill value was greater than the underlying pixel of the exemplar set. In this manner, a "blob" of random complexity and brightness could be added to the exemplar characters. The original public domain code used is reproduced in Appendix B, DevelopersÕ Guide.

5.1.1.3 Pixel Shifts

An artifact of the video taped signal is variation in the absolute x,y position of a character pixel in a digitized frame. A given character may move left-right or up-down by a pixel width or greater within a defined window. Since the character video in the data field overlays a complex video image, edge detection approaches to find the limits of the video character are unworkable, the edges found may as easily belong to the background as to the character. This results in the characters in both the exemplar and data sets "drifting" by a small number of lines or pixels within the bounds of the window.

To compensate for this drift, the exemplar set of characters is shifted by a random number of pixels left/right or up/down during training, thus emulating the drift found in the digitized display. This is depicted in Figure 5-1.

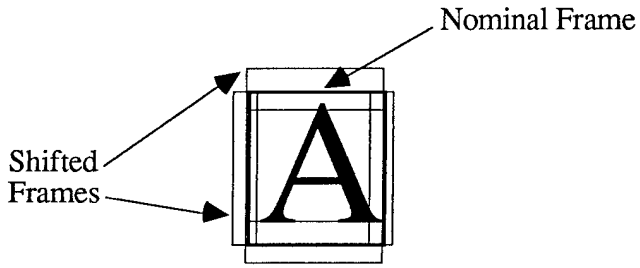


Figure 5-1. Frame Drift / Shift

5.2 Network Architecture

The network architecture was selected based upon both a) assumptions concerning the nature of the data and b) actual empirical data. Factors considered include the need for re-mapping from input to output space, the potential for over-fitting of the exemplar set, and the practical limitations imposed by the DOS operating system, in particular the 640 kilobyte memory limit for the VCRS application. A three-layer network was selected based upon the assumption that re-mapping of the data set is required. No attempt to optimize the number of layers was made.

The input vector size was set by the raw number of pixels in a character frame. A set of 33 exemplars was defined for the target video source, the FLIR imagery previously described. The exemplar size was set at 19 x 31 pixels, tightly framing the data fields. This resulted in an input vector of 589 elements. The option to re-size the field was not used. The 33 characters represent 3 samples for each character type including the blank defined in Table 5-1. The output layer was configured to allow a one-to-one correspondence between output nodes and symbols, resulting in 11 nodes.

Table 5-1. Exemplar Set Symbols

Symbol	Nomenclature
0	Zero
1	One
2	Two
3	Three
4	Four
5	Five
6	Six
7	Seven
8	Eight
9	Nine
	Blank

5.2.1 Hidden Layer Sizing

The size of the hidden layer was determined empirically by the following process:

1) A set of 10 sequential video frames including a total of 40 character or blank fields was defined as a standard test set. These frames were selected as they provided a range of contrast and background complexities, and included samples of all characters.

2) A series of networks were trained against the exemplar set. In this default set, the hidden layer size was varied (typically doubled in successive networks) while all other training parameters were held constant (shift = 2, noise = 64, blobs = 128). The training parameters are discussed in detail in a later section. The number of iterations was not rigidly set. It was observed that system confidence level increased markedly in the first 100 to 200 iterations. System confidence was based on the level (0 - 100) to which each image satisfied several image recognition criteria. The large number of iterations for some runs represented over-night or week-end long training runs. Little improvement in performance was noted after the first 400 to 500 iterations in any of the runs. If no improvement in confidence was observed or an oscillation in confidence was observed, then the run was terminated. Table 5-2 presents the results of this effort.

Table 5-2. Hidden-Node Sizing Trial Results

# Hidden	# Iterations	Confidence	Comments
2	464	0.035	Not evaluated: failed to converge.
4	481	0.266	Not evaluated: failed to converge.
8	863	0.767	38% correct: little improvement in confidence in last 400 cycles
12	Å4000	0.800	20% correct
16	1662	0.906	40% correct
27	1723	0.926	30% correct: Max size under 640k limit.

5.2.1.1 Convergence Behavior:

Both the 2 hidden-node and 4 hidden-node networks exhibited oscillation at very low confidence numbers. No subsequent evaluation of these networks was performed. Their training history is portrayed in figures 5-2 and 5-3. The 8, 12, 16, and 27 hidden-node networks were judged to converge, and were evaluated against the standard set. Their training histories are found in figures 5-4 through 5-7. Attempts to run networks with greater than 27 hidden layer nodes resulted in the application failing due to memory constraints. Currently, the VCRS must operate with the 640k memory limitations imposed by the Microsoft DOS operating system.

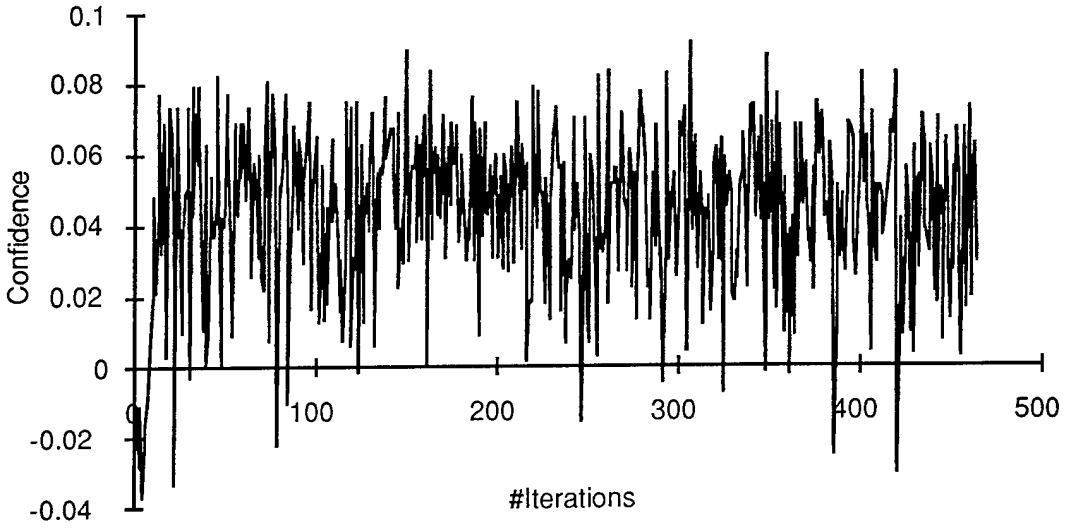


Figure 5-2. Training History for 2 Hidden Nodes

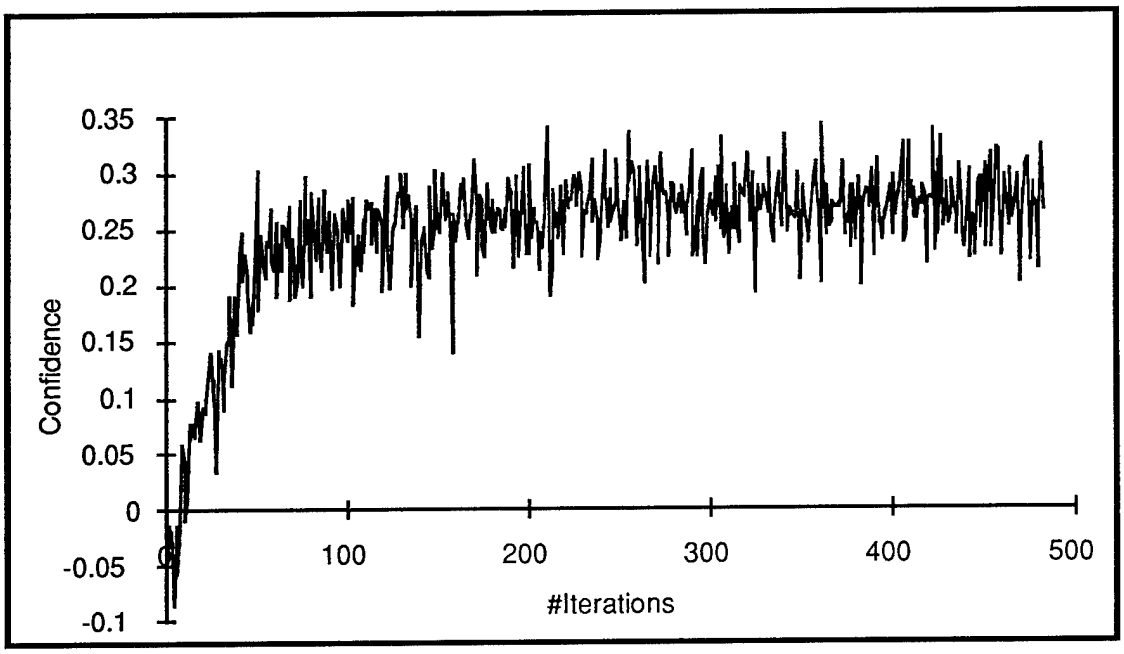


Figure 5-3. Training History for 4 Hidden Nodes

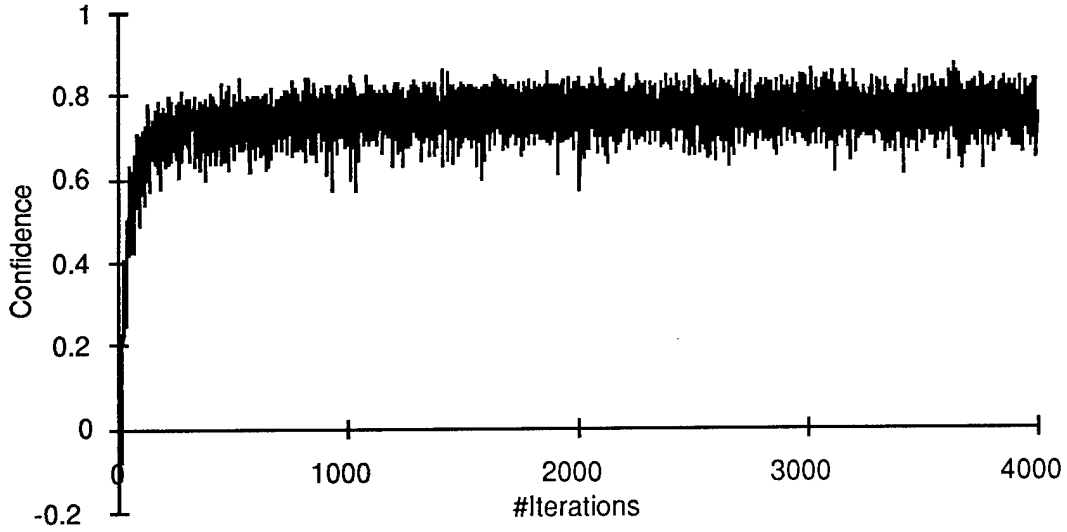


Figure 5-4. Training History for 8 Hidden Nodes

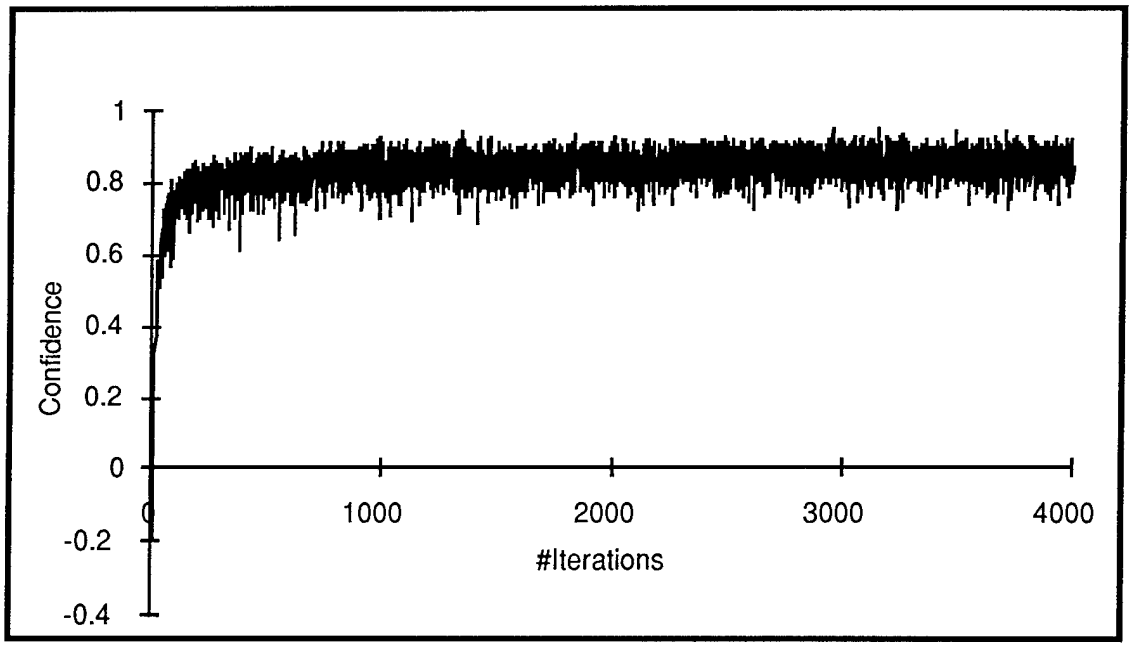


Figure 5-5. Training History for 12 Hidden Nodes

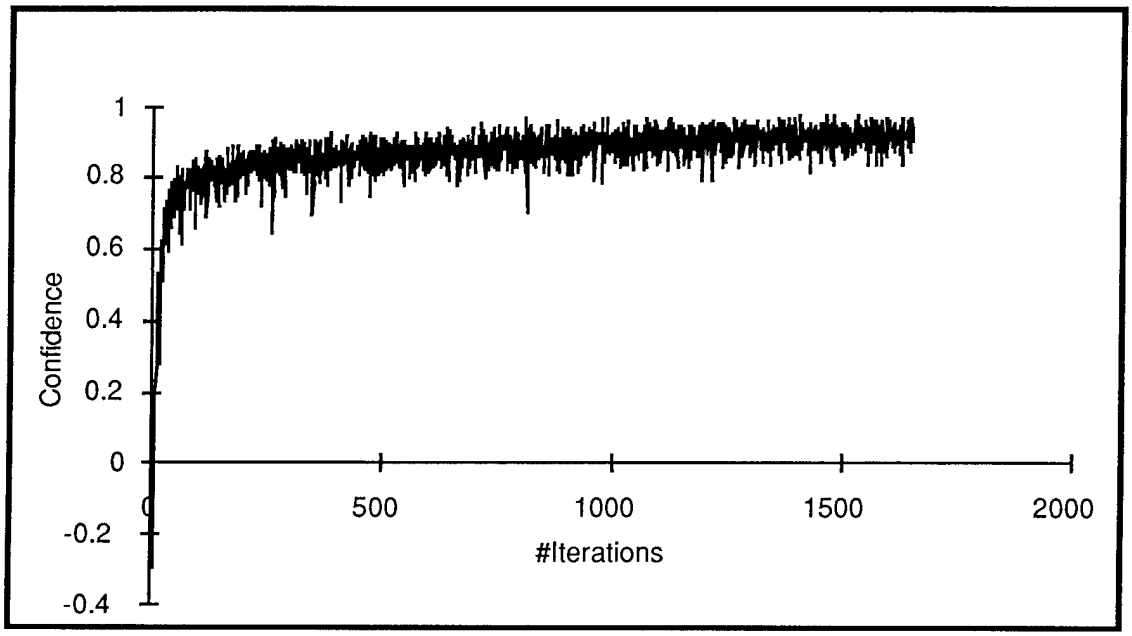


Figure 5-6. Training History for 16 Hidden Nodes

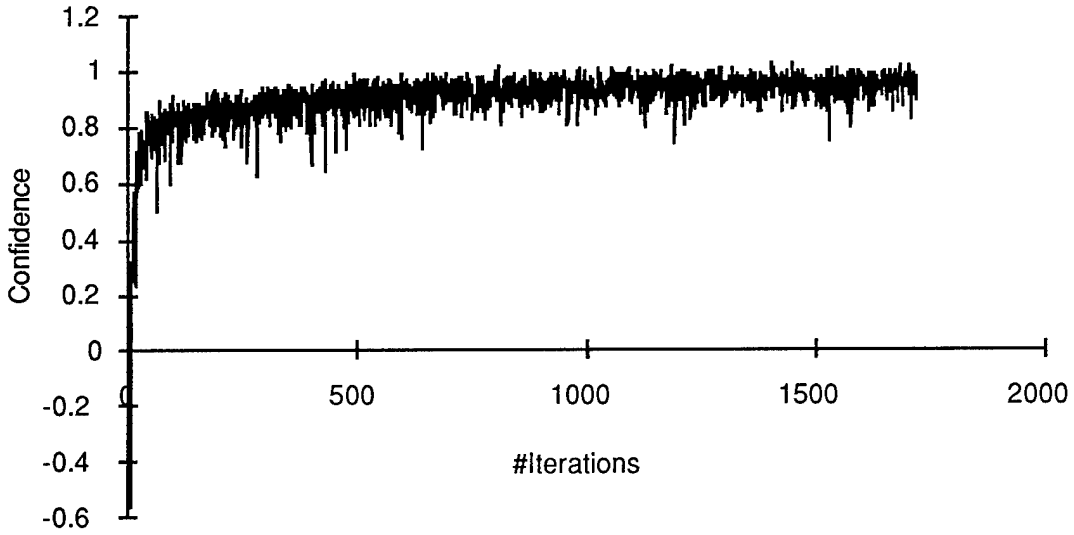


Figure 5-7. Training History for 27 Hidden Nodes

The 27 node network, while achieving a higher confidence in recognizing the exemplar set, was less successful in recognizing the test set. This was believed to be due to over-fitting of the network to the exemplar set, with the smaller hidden-layer size producing better generalization. The large degree of random corruption present in the default training resulted in a scattering of the data about a generally decreasing error in all of the plots. A hidden-layer size of 16 was selected for subsequent testing.

5.2.1.2 Initial Performance Assessment:

After the initial sizing runs, a series of preliminary performance runs was performed in an attempt to bound the effects of the training enhancements available with the VCRS. As previously mentioned, these consist of pixel shifting, random noise addition and "blob" noise addition. A naming convention for network directories was established which described the type of corruption applied during training. This consisted of the standard 8 character DOS file name with the following meaning attached to the fields:

XXYYYZZZ, where:

XX = number of pixels to shift

YYY = Intensity of added blobs

ZZZ = Intensity of random noise

Several 16-hidden-node networks were evaluated using varying combinations of shifts/noise/blobs to further randomize the rather limited exemplar set (33 characters, 3 of each type). All of these networks were trained until confidences were essentially fixed or training was interrupted, and evaluated against the same 10 video frames. The percent correct for each network and systematic errors observed in recognition are summarized in Table 5-3 below. Training history plots are provided in Figures 5-8 through 5-19. Note that although the actual number of iterations varied, only the first 500 iterations for a given network are plotted. The plots are presented in this fashion to allow comparison of convergence behavior, and note that substantial changes beyond 500 iterations did not occur.

Table 5-3 Network Recognition Training Performance

Network Name	**% Correct**	**# Iterations**	**Comments**
00000032	63%	5928	Confused 8/3, 1/b, low contrast
00000064	43%	2755	Confused 8/3, 8/9, low contrast
00000016	63%	587	Confused 8/3, 3/9, low contrast
01000000	53%	665	Confused 8/3, low contrast
02000000	55%	11409	Confused 8/3, 3/9, low contrast
03000000	43&	622	Confused 8/3, low contrast
00016000	58%	609	Confused 8/3, low contrast
00032000	58%	2339	Confused 8/3, low contrast
00064000	65%	677	Confused 8/3, low contrast
00128000	68%	7834	Confused 8/3, low contrast
00192000	68%	2299	Confused 8/3, low contrast
00254000	55%	2884	Confused 8/3, low contrast
02192032	45%	6239	Confused 8/3, low & medium contrast

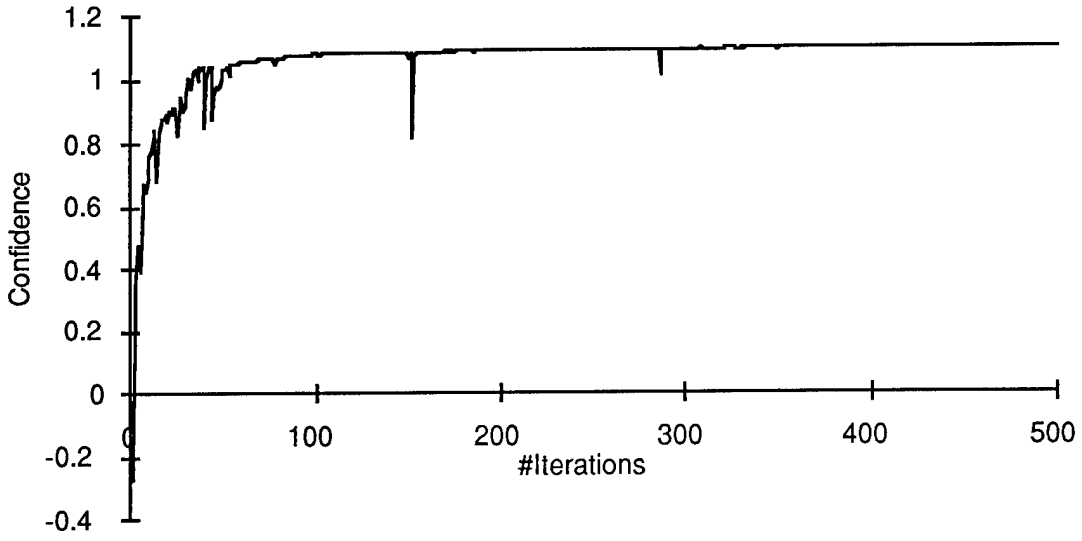


Figure 5-8. Training History for Net 00000016

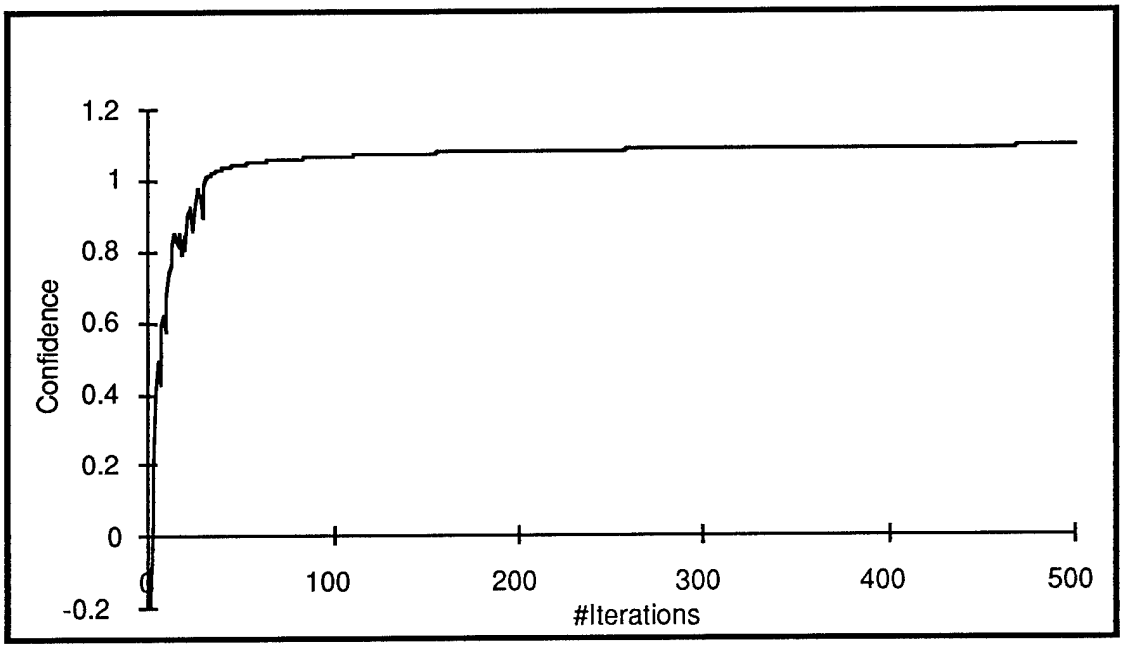


Figure 5-9. Training History for Net 00000032

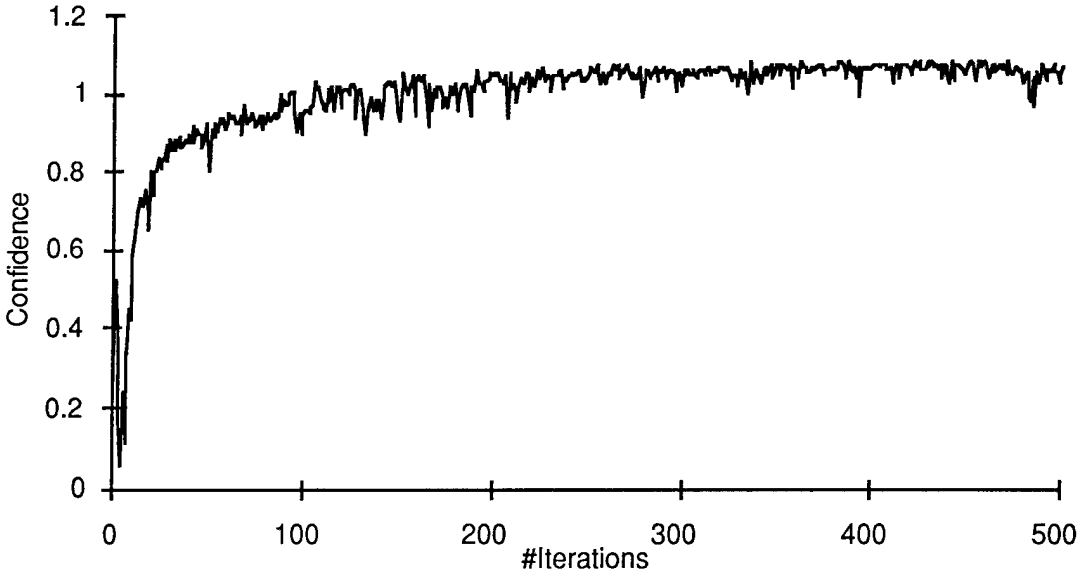


Figure 5-10. Training History for Net 00000064

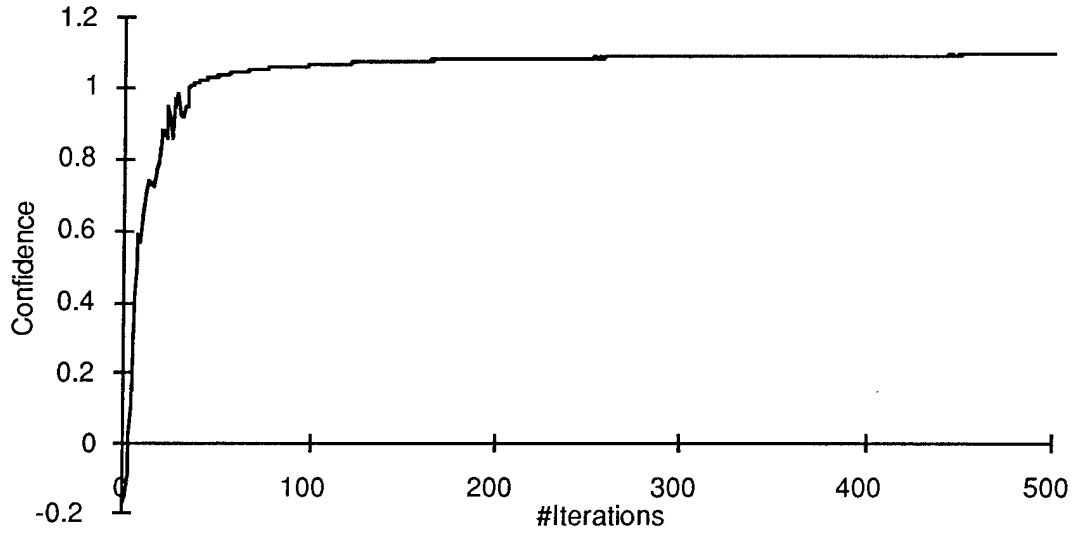


Figure 5-11. Training History for Net 01000000

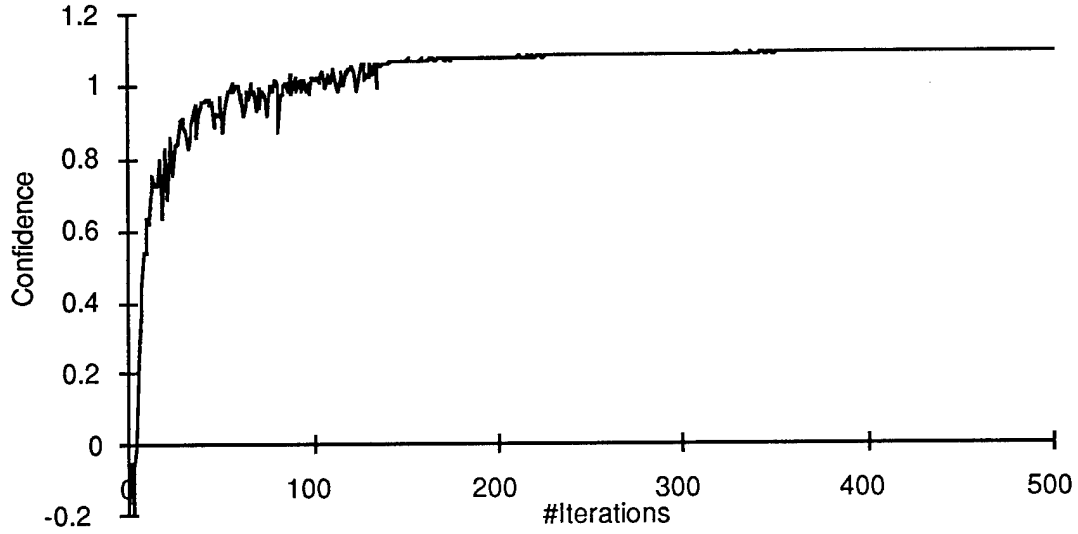


Figure 5-12. Training History for Net 02000000

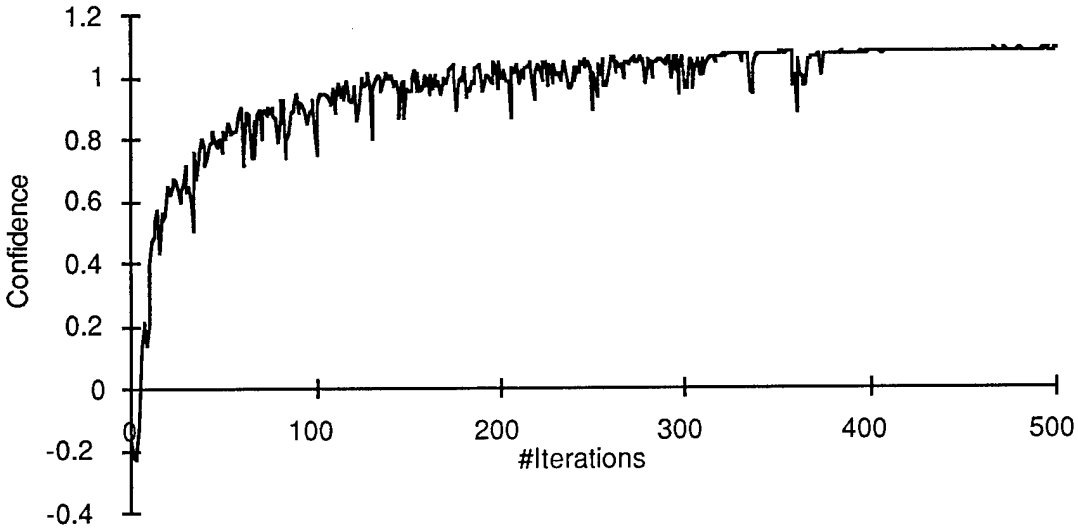


Figure 5-13. Training History for Net 03000000

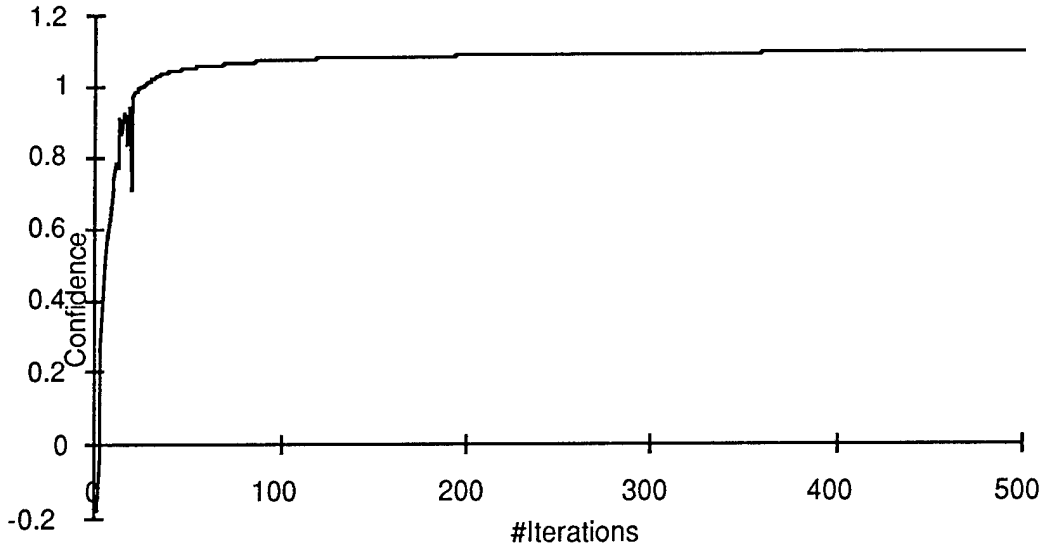


Figure 5-14. Training History for Net 00016000

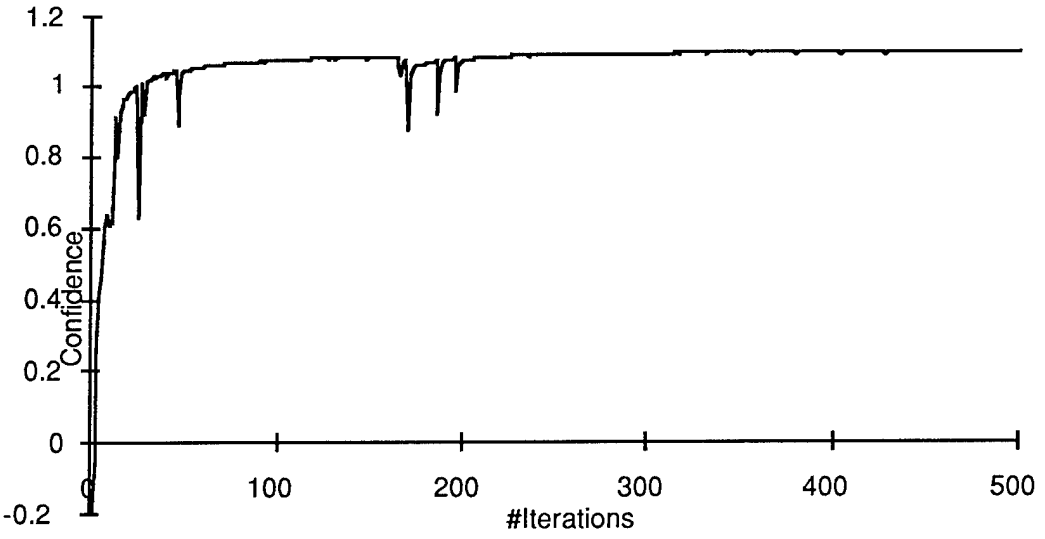


Figure 5-15. Training History for Net 00064000

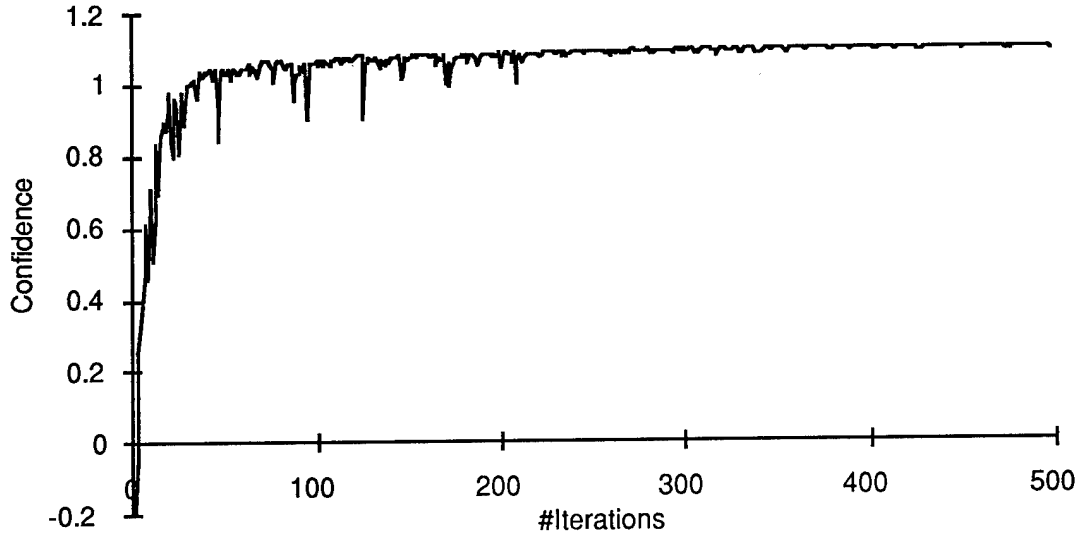


Figure 5-16. Training History for Net 00128000

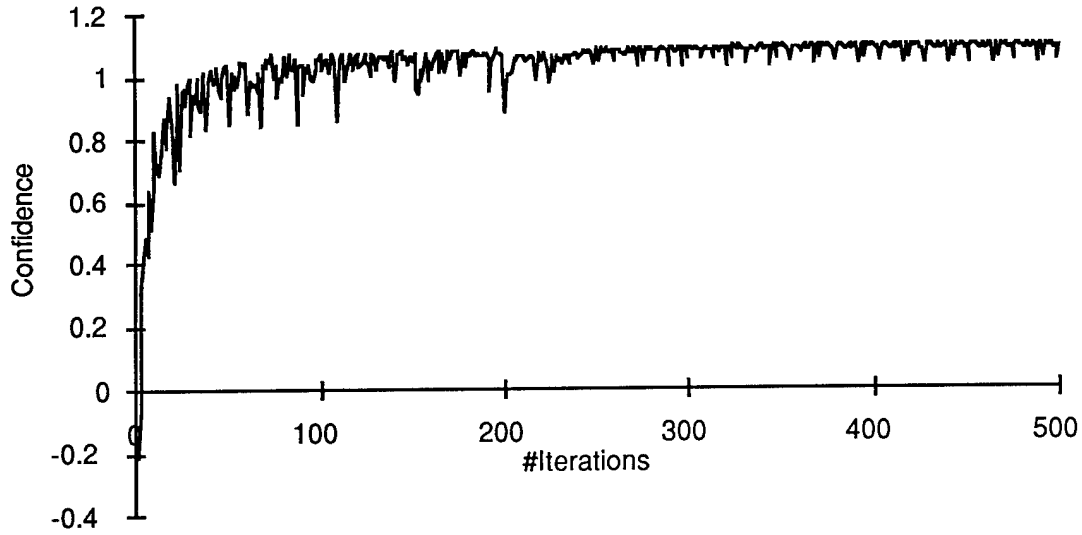


Figure 5-17. Training History for Net 00192000

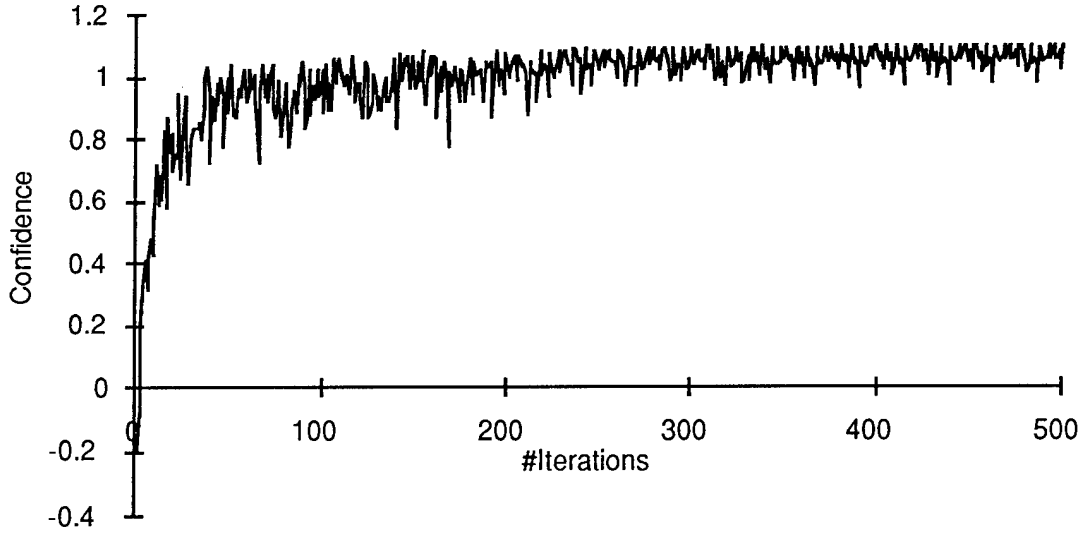


Figure 5-18. Training History for Net 00254000

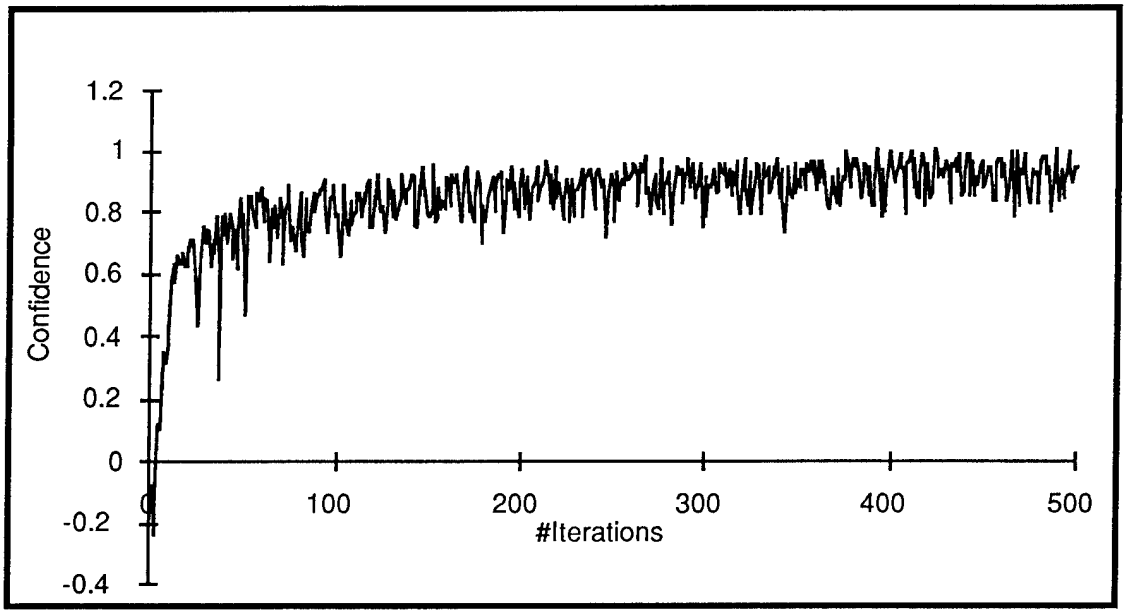


Figure 5-19. Training History for Net 02192032

5.3 Image Pre-processing

In the preliminary runs, several patterns were noted. Several sets of similar characters were confused, most notably 8/3 and 3/9. On close inspection of the frame, it is apparent that all of these characters contain brighter pixels in their horizontal line segments than in their vertical lines. Most likely, this is due to the limited response of the video amplified in the original sensor display leading to a voltage undershoot on lines of a single pixel in horizontal width. This is a common feature of video displays, and must be compensated for if this method is to succeed in general application.

It was previously noted that the FLIR imagery, in general, contains strong horizontal banding in the background of the image due to the line-scanned nature of the image in its formation at the IR detector. This background contributes an inherent structure to the image. In many characters, this banding is nearly as significant as the structure of the video characters.

Finally, in this test sequence it was noted that perfectly legible, but low contrast characters, were poorly recognized. It is believed that this is due to the banding dominating the structure of the character. This is consistent with the large numbers of "b" or blank characters being confused with low contrast characters.

5.3.1 Implementation Approach

The combination of mis-identifications noted here suggests that modest pre-processing of the image might have improved the recognition rate. Pre-processing of the image prior to training and recognition is a logical choice since such processes can be eventually implemented on the video acquisition card (the frame grabber incorporates an on-board digital signal processor and can implement various filtering functions in near-real-time). However, modification of the network software to implement pre-filtering was judged to allow greater flexibility at the expense of real-time performance. This approach, rather than implementing calls to embedded DSP routines, was used to allow for exploration during development.

5.3.2 Pre-processing Modifications

Image pre-processing was implemented using 3 x 3 moving window filters to either smooth or enhance features of the image. The application that implements this variation of the network is named "PREFILT." All of these filters were implemented without any data wrap-around for edges, and thus left an un-processed 1-pixel border around the image. The first filter implemented was a simple median filter to reduce the random noise. Subsequently, 3 x 3 impulse response kernels were added after median filtering to enhance edge features in the image. The input character set, both raw and as modified by the various filters is depicted in Figure 5-20.

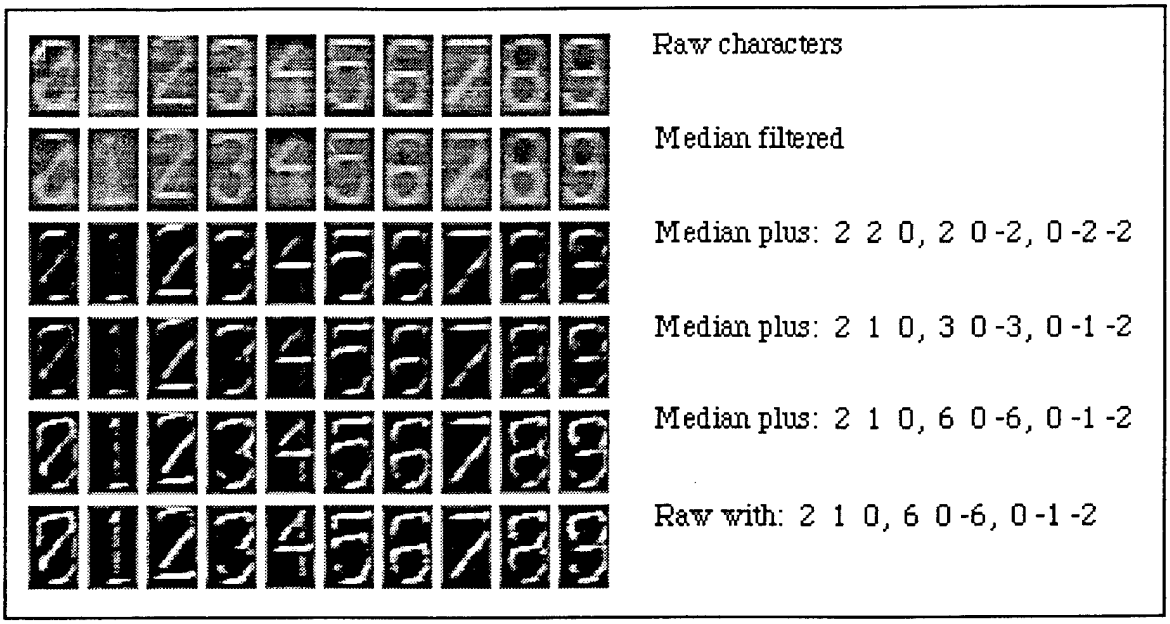


Figure 5-20. Example Character Set Filter Outputs

6.0 RESULTS

6.1 Network Convergence Behavior

As implemented, shifts were not intended for use with these filters as shifting would occur after filtering, leading to prominent edge effects. Blob and noise corruption continued to operate normally. A summary of trial results for image pre-processing appears in Table 6-1 below.

Table 6-1 Network Training Performance.

Network Name	% Correct	# Iterations	Comments
0012800a	55%	905	Median only
0012800b	73%	505	Median, and 2 2 0 impulse response 2 1 -2 0 -2 -2
0012800c	55%	448	Median, and 0 -1 0 impulse response -3 0 3 0 1 0
0012800d	35%	937	Median, and 2 1 0 impulse response 3 0 -3 0 -1 -2
0212800e	60%	2162	Median, and 2 1 0 impulse response 6 1 -6 0 -1 -2
0212800g	50%	579	No Median, 2 1 0 impulse response 6 1 -6 0 -1 -2
0219232a	83% / 78.5%	458	No Median, 2 1 0 impulse response 6 1 -6 0 -1 -2
0219232a	93%	n/a	On images with "normal" content.
0219232a	65%	n/a	On images with "severe" clutter

In Table 6-1, the "% Correct" column typically depicts the percentage of characters identified correctly out of the standard test set of 40 characters (10 images with four-character fields, images A33769 through A33778 on the video disk). However, the entries for 0219232a, which proved to be the best performing network, are unique in that four different values for percent correct are provided. The first entry indicates performance on the standard test set, while the second is for 200 characters (50 four-character fields on frames A33750 through A33799). This latter set was then separately scored as "normal" images, where backgrounds were uncluttered or exhibited low frequency noise, and images with "severe" clutter. In general, the latter set of frames included FLIR imagery with a shipyard background, where the many masts, superstructures and waterlines filled the background with high contrast linear shapes readily

enhanced by the impulse response filter. Several of the characters in these fields were difficult for the experimenter to unambiguously identify.

Trial 0012800a, shown in Table 6-1 above and Figure 6-1 below, includes a 3 x 3 median prefilter in addition to the 128-level blob noise corruption of the exemplar set. This configuration was selected in an attempt to improve the network performance by reducing the impact of the FLIR banding present in most images through the prefilter, and the addition of blob noise to reduce local minimum effects during training. The convergence behavior displayed, however, was unremarkable.

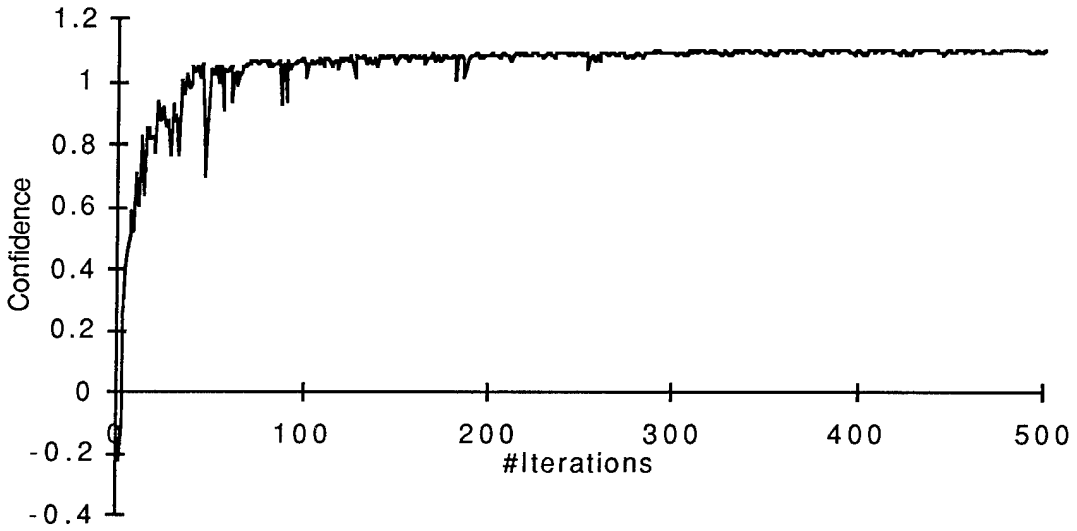


Figure 6-1: Training History for Net 00128000a

Trial 0012800b, depicted in Figure 6-2 below, included a diagonal edge detection function to attempt to enhance the unique diagonal line elements of the features. Unlike the use of edge detection to determine the location of the character within a large field where noise can cause a false determination of location, edge detection, here, was performed only on the sampled window. This limited area was assumed to contain both the character and background noise. The edge detection filter can be tailored to preferentially enhance the features of the desired characters while changes in noise are assumed to be random. While the filter applied in this trial further confused the distinction between 3/8, 3/8, 8/9, it was capable of greatly improving the network performance in low contrast characters to the degree that over 70% of the test character set was correctly identified. Once again, the behavior of the network was unremarkable.

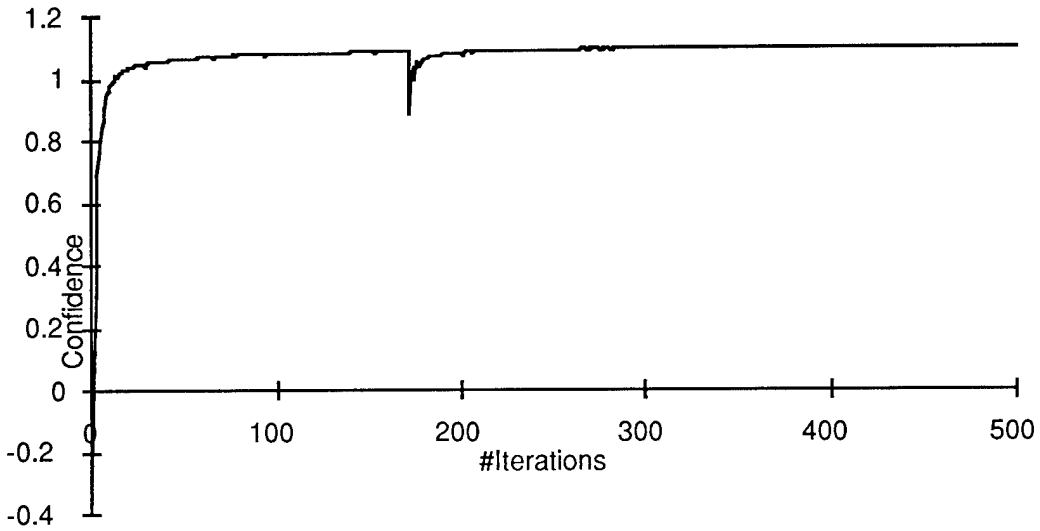


Figure 6-2. Training History for Net 00128000b

Trial 0012800c, depicted in Figure 6-3 below, was an attempt to improve the distinction between the 3/8, 3/9, 8/9 confusers by weighting the edge enhancement of vertical line elements over horizontal. This was not successful. Observation of the character set found that the diagonal elements of these characters were essential in defining the closed/open curves that distinguish them. The training history plot depicted indicates that the network became trapped in a local minimum early in training, escaping only after nearly 300 iterations. The maximum confidence achieved was lower than similar nets, indicating that a global solution was never achieved for this network.

This highlights the nature of random processes in the network training processes, and the need to train to a confidence limit rather than a fixed number of iterations. It is unclear if further training would further improve the network performance, but it is assumed to be likely since a random blob corruption of the exemplar was enabled.

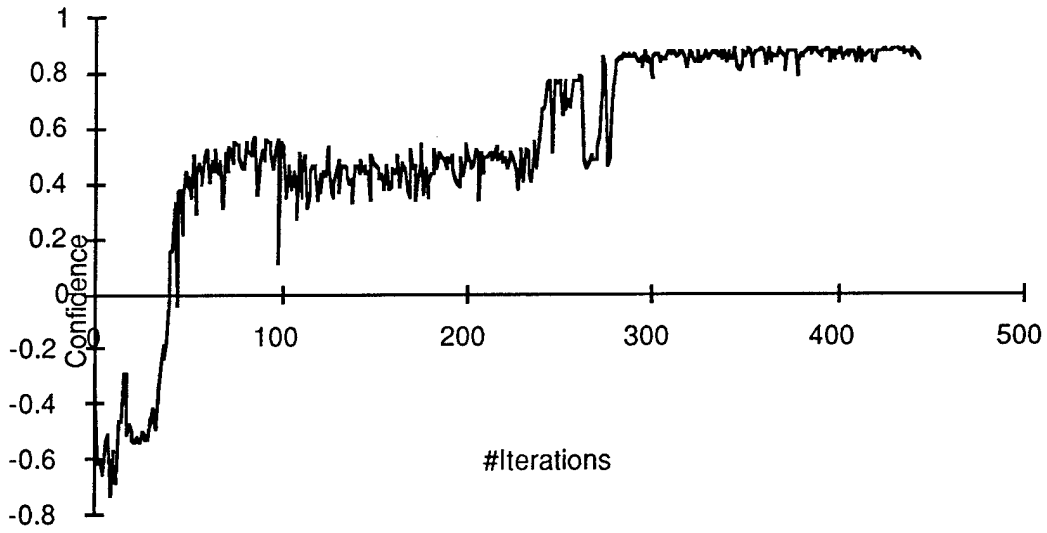


Figure 6-3. Training History for Net 00128000c

Trial 0012800d, Figure 6-4 below, attempted to improve upon the performance of the previous trials by adding a diagonal element back into the impulse response matrix. The training history in this case is unremarkable.

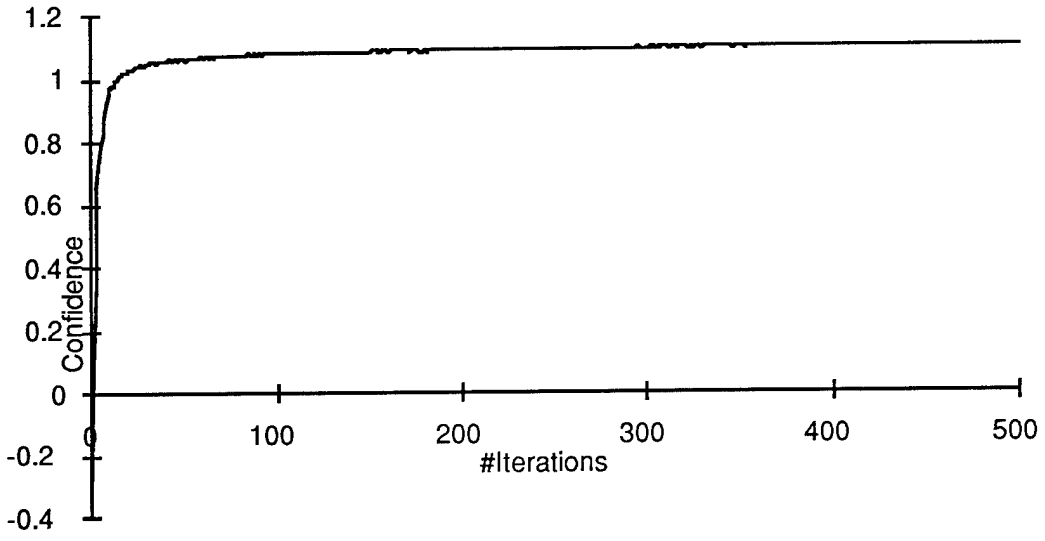


Figure 6-4. Training History for Net 00128000d

Trial 0212800e, shown in Figure 6-5, increased the amount of enhancement applied to vertical elements in a further attempt to compensate for weak video while adding a 2 pixel random shift. The increased randomization induced by the pixel shift is exhibited in the low-level "roughness" continuing throughout the plot.

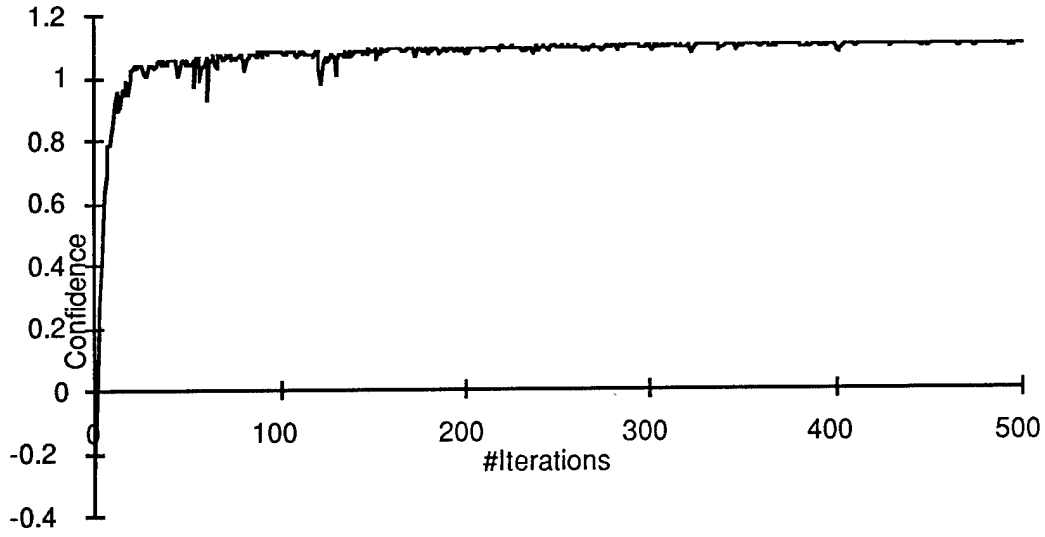


Figure 6-5. Training History for Net 00128000e

A variation of trial 0212800e had the median filter disabled and was logged as trial 0212800g. Trial 0212800f was skipped due to a directory error. Although trial 0212800g resulted in less successful recognition of the test set, the visual appearance of the strongly enhanced images without the initial median filter appeared robust and was, apparently, "easy" to recognize.

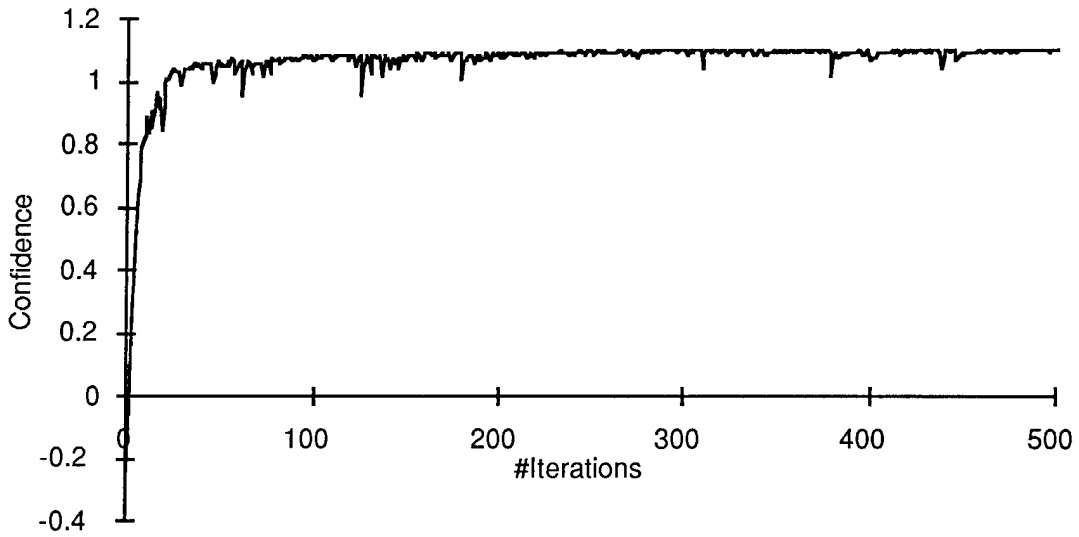


Figure 6-6. Training History for Net 00128000g

Trial 0219232a, Figure 6-7, incorporated the more robust filtering used in trials e and g, along with the levels of increased random noise and blob corruption found to perform best in isolation. This was done in an attempt to produce increased generalization of the trained network. This approach yielded the highest recognition rate of any trial. The method was largely insensitive to most contrast changes, even when such changes divided the character frame horizontally. This is due to the weighting of the edge enhancement filtering. The approach was understandably sensitive to the severe background clutter present in many of the images. However, some characters that did not have high levels of clutter, especially "1," were mis-identified and had no obvious corruption. This is indicative of a failure to reach a truly general solution for the mis-identified characters during training. Addressing this issue should result in further increases in recognition rate.

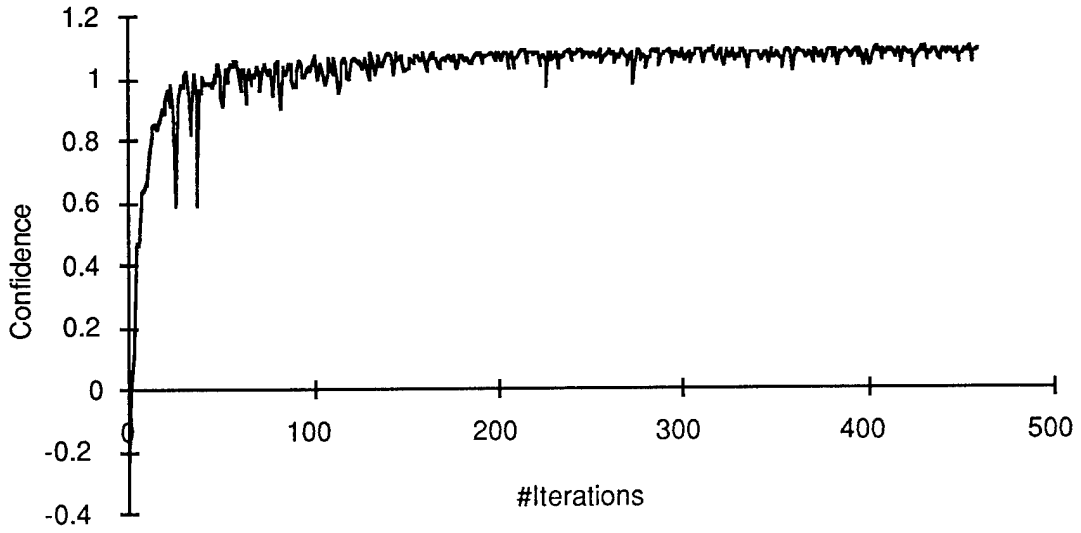


Figure 6-7. Training History for Net 02192032a

6.2 System Performance

Training with even the limited number of exemplars present proved to be time consuming. Each "iteration" depicted in the chart above accounts for 10 passes through the exemplar set before updating the network. The time required for this ranged from 31 seconds per iteration (10 passes) for network 0000016 which incorporated only modest random noise, to 62 seconds per iteration for network 0219232a, which included pre-filtering, blob noise, random noise, and pixel shifts. This led to training periods of, minimally, several hours for each network, with training runs frequently extending overnight. No data was obtained on recognition performance, but is typically dominated by computer-video frame acquisition time rather than network performance.

This page intentionally left blank.

7.0 FUTURE PLANS

7.1 Software Enhancement

The current software was designed during the development phase of the program. The plan for future development was to add additional software in order to give analysts the ability to use the system with minimum computer and systems knowledge. The current system is configured with all the development options in the training systems. Any significant improvements require that the 640kB memory limitation in DOS be eliminated, either through the use of extended memory techniques or use of a less limited operating system. The first major enhancement will be to develop a software overlay for the network training module. The training module will give less experienced users the option of default values for the training. The variables include all the values in Appendix A, Figure A-4 which would otherwise be set by the software.

The second major enhancement will be to add interfaces to existing and planned flight test software systems. AL/CFHI is currently designing system interfaces to evaluate the Test Planning, Analysis and Evaluation System (PAES). These interfaces will significantly increase the useability of both the VCRS system and the flight test system to which it is interfaced.

Further enhancements to the character recognition processes can be implemented to improve performance in high clutter backgrounds. The current processes achieve useful recognition rates only in low to moderate clutter scenes such as those encountered in air-to-air and maritime situations. Recognition rates remain unsatisfactory for complex clutter backgrounds likely to be encountered in air-to-ground scenarios. Several candidate techniques can be applied to improve this performance.

- Variations in training: Observation of the errors encountered in the current approach indicate a failure to reach a generalized solution during training of the neural network. This problem has been addressed frequently in the literature. One alternate approach, excess output training, has been suggested as a design remedy by Yeong-Ho, Y. and Simmons, R. (1990). In this approach, output nodes are assigned that correspond not only to the desired characters, but to classes represented by several characters. For example, the characters 6, 8, 9, and 0 are members of the class of characters with closed curves, while 2, 4, 7, and 0 are characters (in the set used in the trials) that contain diagonal lines. By training the neural net to represent classes of characters as well as individual characters, the network is forced to generalize its internal representation of the data. These excess outputs are later deleted from the trained network.

- Hierarchical network architecture: Hierarchical organizations of networks have been previously used in conventional optical character recognition tasks by dividing the target character set into specialized sub-classes, Sabourne, & Mitiche, (1992). A variation of this approach can be applied to the current problem to deal with the

high levels of background clutter. While the current network attempts to perform recognition based on a single network operating on a single input vector. An alternate approach is the use of multiple networks operating independently upon several variations of the original input vector. A second, rule-based stage then combines the separate results of the networks to yield improved recognition. For the text recognition task, this approach can capitalize upon the variations in horizontal and vertical symmetry present in the character set, assuming noise remains random. For example, a character window can be manipulated by mirroring the character about its vertical and horizontal centerlines to yield left/left, right/right, top/top, and bottom/bottom symmetry pairs in addition to the original image. This process would yield five separate input vectors for recognition.

The power of this approach can be understood if the symmetry pairs yielded from characters 6, 8, and 9, are examined. These characters were often confused in the current trials. Comparison of top/top, bottom/bottom pairs mirrored about the centerline yields distinctly different patterns for each character.

The last major software enhancement is to add a voice command interface to the system.

7.2 Design of Prototype System

If the VCRS is to be produced for its intended use, then the configuration must be changed significantly. The development configuration is somewhat piecemeal for the convenience of switching out hardware components. This optimizes system design for purposes of hardware upgrades and enhancements. Any future production system will be housed in a single hardware box. The single hardware configuration will incorporate all necessary hardware and software components as well as display devices and interface connections.

Two configurations hold the greatest promise for design and manufacturing. The first configuration will be for the desktop and for post processing of data collected in aircraft and other applications. The second configuration will be for airborne use to do real-time analysis of video data. The flight system configuration will require aircraft data bus connectors in addition to the standard connectors used in the desktop configuration. Also, the flight configuration will require EMI and vibration certification.

8.0 CONCLUSIONS

The development of the VCRS has demonstrated the feasibility of developing an automatic character recognition system. Overall, the VCRS achieved a relatively high character recognition rate. More specifically, the inclusion of a diagonal edge detection function was successful in improving network performance on low contrast characters to better than 70% correct identification but further confused the distinction between the characters 3, 8 and 9. Weighting the edge enhancment of vertical line elements over horizontal line elements did not improve the networkÕs ability to distinguish between the 3, 8 and 9 characters.

The attempt to increase generalization of the trained network through the use of more robust filtering techniques (vertical element enhancement, 2 pixel random shift & disabling of median filter) and increased levels of random noise and blob corruption produced the highest recognition rates. This method was largely insensitive to most contrast changes but, was understandably sensitive to high background clutter. Some characters that had no obvious corruption, especially the Ò1,Ó were misidentified due to a failure to reach a truly general solution during training. Training with even limited numbers of exemplars proved to be time consuming. Time required for each ÒiterationÓ ranged from 31-62 seconds. Training periods typically lasted more than several hours.

Despite the advances made by the initial VCRS development, the VCRS could only achieve a 75 - 80% solution. Further improvement of the VCRS cannot be achieved until a more robust character recognition system is developed that is tolerant of the varying backgrounds and noise levels present in the video environment. To this end, planned enhancements to the VCRS include 1) development of a software overlay for the network training module, 2) adding interfaces to existing and planned flight test software systems (e.g., Test PAES), 3) adding a voice command interface and 4) further enhancements to the character recognition process; especially for characters in high clutter backgrounds. One suggestion for increasing performance in high clutter backgrounds is to increase the probability of reaching a generalized solution by implementing a variation in training called Òexcess output trainingÓ (Yeong-Ho and Simmons, 1990). Another is to implement a hierarchical network architecture which divides the target character set into specialized sub-classes (Sabourne & Mitiche, 1992).

The VCRS program has had two significant accomplishments. The first is the development of an imported tool for evaluating a wide class of video data. This tool will significantly reduce manpower requirements for analysis of all video with embedded character data. The second is a major advance in neural network theory. The use of artificially generated non-linear noise injection was demonstrated in conjunction with standard random noise injection techniques. The non-linear "blob" noise simulates background clutter more generally than previously available models. The use of complex non-linear models of systems and noise behavior will increase the useability of neural networks both in high noise environments and in problems with limited, available example data.

This page intentionally left blank.

BIBLIOGRAPHY

Eglowstein, H., (1994). "Due Recognition for OCR," Byte, pp 145-148.

Hatley, D., and Pribhai, I., (1988). Strategies for Real-Time System Specification.

Moray, N., (1980). *Human information processing and supervisory control* (Tech. Rep. NR 196-132). Cambridge, Mass.: Massachusetts Institute of Technology.

Moray, N., (1984). Attention to dynamic displays in man-machine systems. In R. Parasuraman & D.R. Davies (Eds.), *Varieties of attention*. Orlando, Fl.: Academic.

Rumelhart and McClelland, (1986). Parallel Distributed Processing, MIT Press.

Sabourne and Mitiche, (1992). "Optical Character Recognition by a Neural Network," Neural Networks Vol. 5, pp. 843-852.

SCB-100N Video Tutorial, (1987). The Grass Valley Group, Inc., p. 3.

Wasserman, Philip D., (1989), Neural Computing Theory and Practice, by Van Nostrand Reinhold, New York.

Yeong-Ho, Y. and Simmons, R, (1990). "Extra Output Biased Learning," Proceedings of the International Joint Conference on Neural Networks.

This page intentionally left blank.

ABBREVIATIONS AND ACRONYMS

AL	Armstrong Laboratory
AM-CLR	Color Aquisition Module
ART	Adaptive Resonance Theory
AZ	Azimuth
BP	Back-propagation
CPU	Central Processing Unit
DFD	Data Flow Diagram
DOS	Disk operating system
DSP	Digital Signal Processor
DT & E	Development, Test & Evaluation
EL	Elevation
FLIR	Forward Looking Infra-red
HUD	Heads-up Display
IA	Image Analyst
IR	Infra-red
LLLTV	Low Level Light TV
MB	Megabyte
MFG	Modular Frame Grabber
OCR	Optical Character Recognition
PC	Personal Computer
Pixels	Picture Elements
Q, T & E	Qualification, Test & Evaluation
SAC	Strategic Air Command
S-VHS	Super VHS
TBC	Time-based connection
T, D & E	Tactics, Development & Evaluation
VCRS	Video Character Recognition System
VTR	Video Tape Recorder/Player

This page intentionally left blank.

GLOSSARY

Best Confidence - The highest confidence level encountered during a set or sequence of different input images and system recognition.

Best Count - The number of input images which were most often recognized correctly.

Exemplar - An individual example or instance of a particular character which is used by the character recognition system as an example to represent that class of input.

Field Editor - The section of the character recognition system which allows the user to define the parts (fields) of an input video screen which contain characters which the user wishes to recognize.

Frame Grabber - An electronic and software system for capturing the data on a single frame of video input and storing it in memory for later processing.

Net Trainer - The computer algorithm and systems used to develop an operating recognition algorithm customized for the particular class of input data currently of interest to the user.

Neural Network - A computer or computer program whose operations are designed and based on the functioning of neurons in the human brain.

Worst Confidence - The lowest confidence level encountered during a set or sequence of different input images and system recognitions.

This page intentionally left blank.

APPENDIX A
VCRS USER MANUAL

A1 SCOPE

This document defines the VCRS user interface and the process description specific to creation, training and evaluation of an artificial neural network. The system configuration necessary to operate the VCRS tool is described. This user manual does not describe the video recording process, nor the modification of the VCRS software configuration.

A2 PURPOSE

The purpose of the user manual is to familiarize the VCRS user with the graphical user interface (GUI) peculiar to the application of the VCRS tool. Specifically, this document describes the processes and related procedures specific to exemplar data set creation, neural net training, and evaluation of neural net performance in video character recognition.

A3 INTRODUCTION

This section defines the system hardware configuration and the VCRS software application modules.

A3.1 HARDWARE CONFIGURATION

The following paragraphs define the computer system and peripheral components necessary to configure a VCRS system. There are three principal components of the VCRS system: 1) the computer, 2) the optical disc player, and 3) the video frame grabber/DSP.

A3.1.1 Computer

The VCRS application software is configured on an IBM-compatible personal computer (PC). The PC CPU is based on the Intel 486DX microprocessor. The operating system is Microsoft DOS, version 5.0. System user memory consists of 32 MBytes RAM and a 213 MByte secondary storage drive. Other peripherals include a 3.5 inch flexible diskette drive, a 101 keyboard, and mouse.

A3.1.2 Video Frame Grabber/Digital Signal Processor (DSP)

The video frame grabber is a PC-based card which hosts a modular frame grabber function integrated with a digital signal processor. The DSP is a Texas Instrument TMS32020. The card, a model MFG AM-CLR, was developed by Image Technology, Inc. The card was configured to operate in a two monitor mode. One monitor (SVGA) is dedicated to the DOS system interface, while the other monitor is dedicated to the VCRS graphical user interface. The VCRS monitor is a high resolution, 1024 x 1024 screen with multisync scan capability. The VCRS monitor is a Diamond Scan model, manufactured by Mitsubishi.

A3.1.3 Optical Disc Player

The optical disc recorder/player facilitates the playback of the video source in a random manner. The video format supports an S-VHS format with an RS-170A timing specification, common to the video industry. The optical disc media supports up to 54,000 video frames. In addition, a communications interface can support multiple frame playback under computer control. Note, a multi-frame playback mode was not implemented since data recognition performance was collected on a frame-by-frame basis controlled by the optical disc remote control. The optical disc recorder/player is a model TQ-3031F, manufactured by Panasonic.

A4 START UP AND INSTALLATION

Subsequent to booting DOS, create a VCRS directory. Copy the following files from the 3.5 floppy diskette onto the Hard Disk Drive (HDD).

1) prefilt.exe
2) trial.frm
3) net.exe

Note, each trained neural net requires a separate directory to store net data files; otherwise existing data files are overwritten as a new trainer session. Execute the NET.EXE to start the VCRS application. Upon execution, the MAIN Screen is activated. This screen is the entry point to the VCRS GUI. Figure A-1, VCRS Structure, displays the hierarchical structure of VCRS screens and their related functions.

A5 VCRS USER INTERFACE

The MAIN screen displays the command line field at the bottom of the screen, with the following command lines.

a. FieldEdit — activates the FIELD EDITOR Screen.
b. ExEdit — activates the EXEMPLAR SET EDITOR Screen.
c. MakeTrain — activates the EXEMPLAR SET SIZING Screen.
d. NetTrainer — activates the NET TRAINER Screen.
e. EvalNet — activates the EVALUATE Screen.
f. Exit — terminates command line/session.

Note, the command lines displayed at the bottom of the screen are dependent upon the state of exemplar set creation, neural net training, and evaluation. Hence, if the user starts the VCRS application with no exemplar set having been previously created, then only the FIELDEDIT and the EXIT command lines will be present.

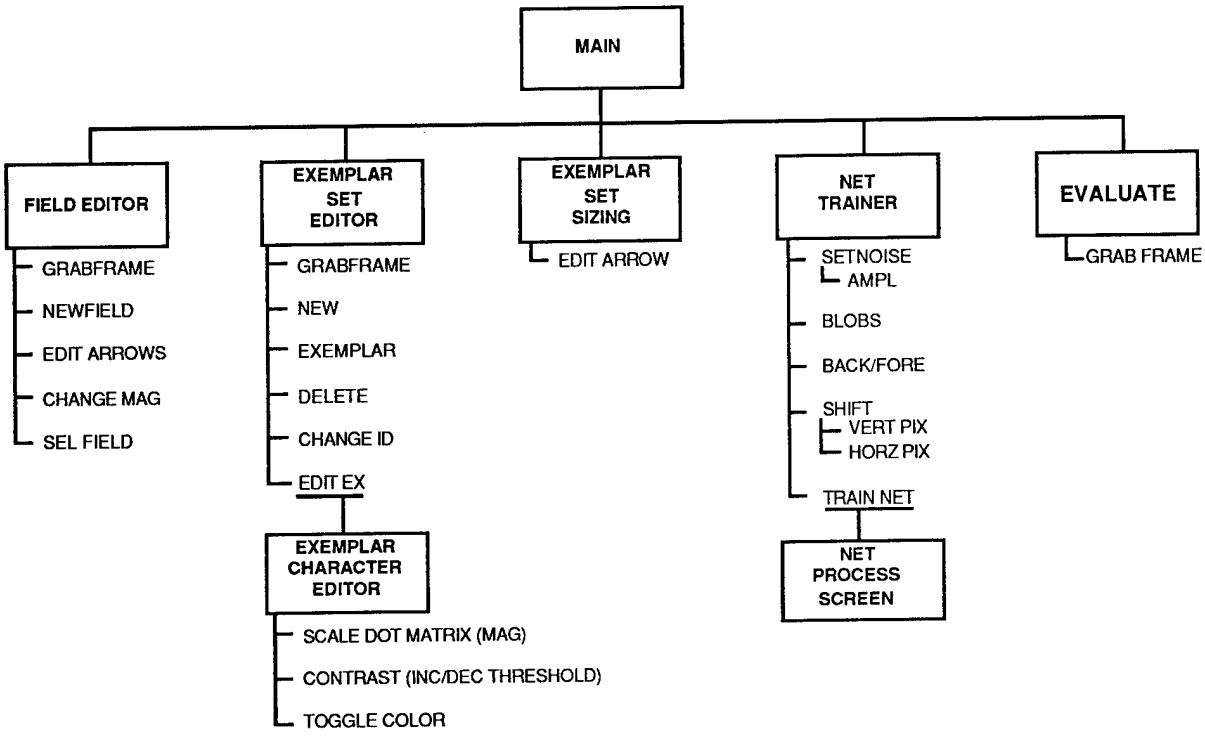


Figure A-1. VCRS Structure

The VCRS user interface provides a common method of execution for all commands. The mouse button is used to select/activate a screen command. Command execution requiring user input parameters will appear as a MESSAGE query. Parameters will be entered by the user with the keyboard interface. The BACK command is used to page back to the previous screen.

Figure A-2, VCRS Environment, and Figure A-3, VCRS Process, provide an overview of the VCRS process in the context of screens and the user interface. The process begins by capturing a selected frame of video from the source. The user applies the FIELD EDITOR screen functions to digitize a video frame and define the character fields within the video. After field editing the video, the user enters the EXEMPLAR SET EDITOR screen. The exemplar edit interface supports the creation of an exemplar set from within the defined character fields across multiple frames of the video source. The EXEMPLAR SET EDITOR screen supports user modification of the pixel matrix size of an exemplar set before submittal to the neural net. The NET TRAINER screen supports application of various formats of corruption to the exemplar set. Methods of corruption include additive random noise and structured noise (BLOBS), character shifting, and foreground and background luminance values. The EVALUATE screen supports the submittal of selected video characters to the trained neural net for the purpose of character recognition.

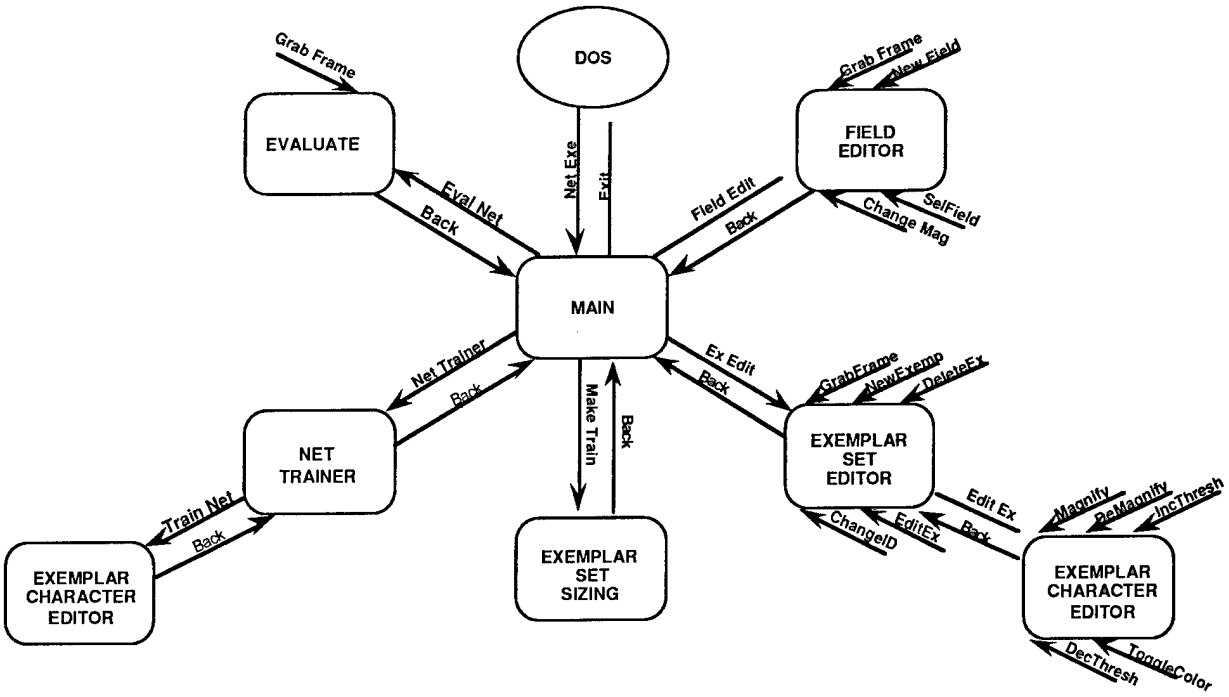


Figure A- 2. VCRS Environment

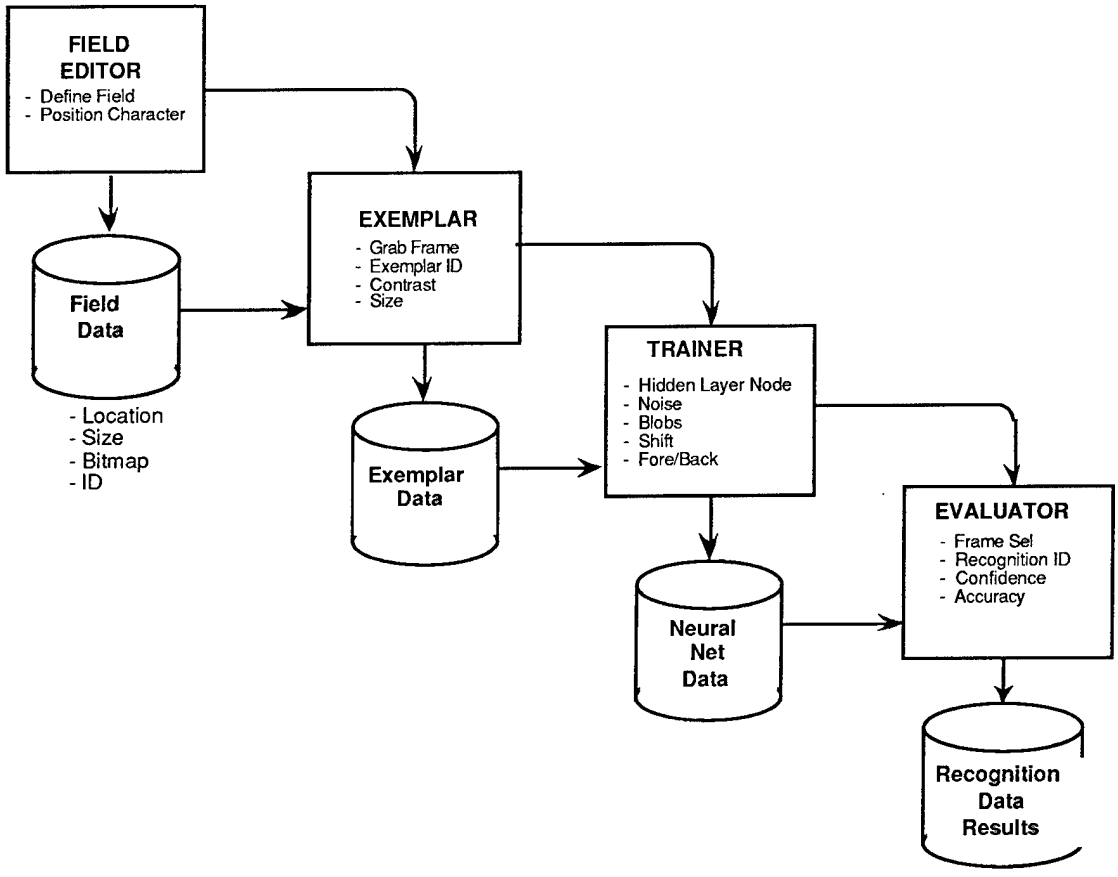


Figure A-3. VCRS Process

The Evaluate process applies the trained neural net to compute the output weights of video characters within the selected video frame. The neural net determines the ID of each character and the confidence level of the identification. The following paragraphs describe user procedures within the context of each VCRS screen.

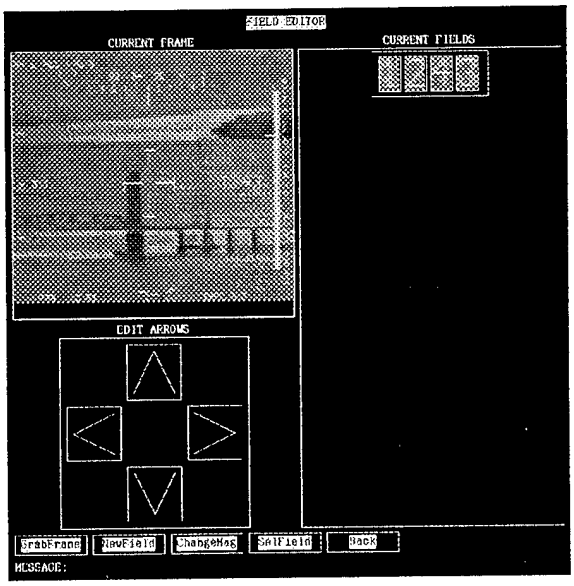


Figure A-4. Field_Editor

A5.1 FIELD EDITOR

From the MAIN screen, select the FIELD EDIT command to activate the FIELD EDITOR screen. Upon activation of the FIELD EDITOR screen, select the video frame of interest from the video playback component. Execute the GRABFRAME command. The GRABFRAME command digitizes the selected video frame. The digitized video is displayed in the CURRENT FRAME window. Execute the NEWFIELD command. Use the mouse button to perform a click and drag action in order to define the character field in the CURRENT FRAME window. Enter the number of characters in the selected field with the keyboard. The CURRENT FIELDS window displays the bit map of each character bounded by the NEWFIELD. Execute the CHANGE MAG function to scale the size of characters. A factor of 2x is the default magnification character size value. Enter other magnification values through the keyboard. Use the mouse button to apply EDIT KEYS to center each character within its field. Upon completion of character field editing, select the BACK command with the mouse to return to the MAIN screen. Enter 'yes' to the "save your field" data query.

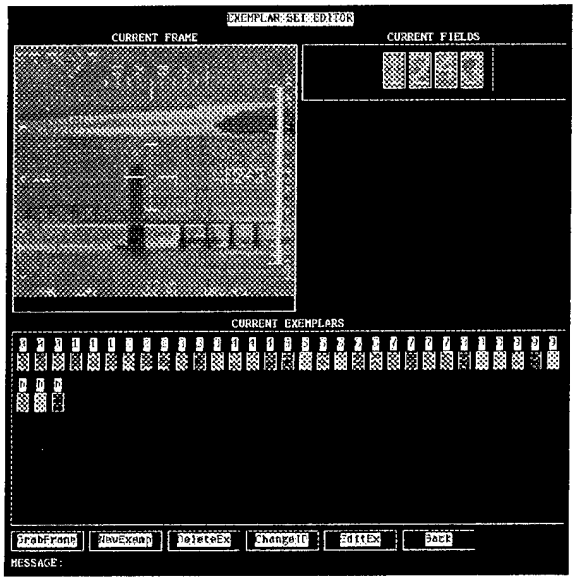


Figure A-5. Exemplar Set Editor

A5.2 EXEMPLAR SET EDITOR

From the MAIN screen, select the EX EDIT command to activate the EXEMPLAR SET EDITOR screen. Execute the GRABFRAME command to digitize a different frame. Execute the NEW EXEMPLAR command. Use the mouse to select a character from CURRENT FIELDS window. The selected character will be added to the exemplar set displayed in the CURRENT EXEMPLARS window. Enter the ID of the selected character with the keyboard. The CURRENT EXEMPLARS window is now updated with the new exemplar character. Perform exemplar set revisions, such as deletion and/or identification, with the execution of the DELETEEX and the CHANGEID commands.

Repeat the NEW EXEMPLAR command process to select other characters from the CURRENT FIELDS window. Execute the GRABFRAME command as other video frames are selected. Repeat the process to identify a new exemplar and append to the current exemplar set.

The EDITEX command is used to modify the contrast of a selected exemplar. Upon execution of EDITEX, select the exemplar character to modify. The EXEMPLAR CHARACTER EDITOR screen, seen in Figure A-6 below, becomes active with the selected exemplar displayed. This screen provides a user interface to modify the contrast of individual exemplars. Execute the MAGNIFY/DEMAGNIFY command to scale the size of the selected exemplar for readability of character before editing. Next, execute the INCTHRESH/DECTHRESH commands to modify the contrast of the exemplar dot matrix field. Execute the BACK command to return to the previous screen.

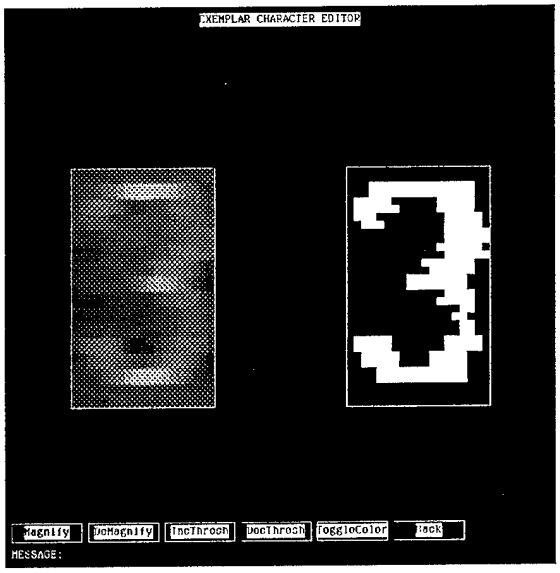


Figure A-6. Exemplar Character Editor

A5.3 EXEMPLAR SET SIZING

From the MAIN Screen, execute the MAKE TRAINER command to activate the EXEMPLAR SIZING screen. See Figure A-7, Exemplar Set Sizing screen. The EXEMPLAR SET SIZING screen provides the user an interface to modify the size of exemplar set displayed within the CURRENT EXEMPLARS window. Select the exemplar set with the mouse. Select EDIT ARROWS with the mouse and use the button to modify, i.e., increase and/or decrease, the pixel matrix size of the entire set. After modification of exemplar, execute the BACK command and save the training set. The exemplar set is now defined and ready to be submitted for neural network training.

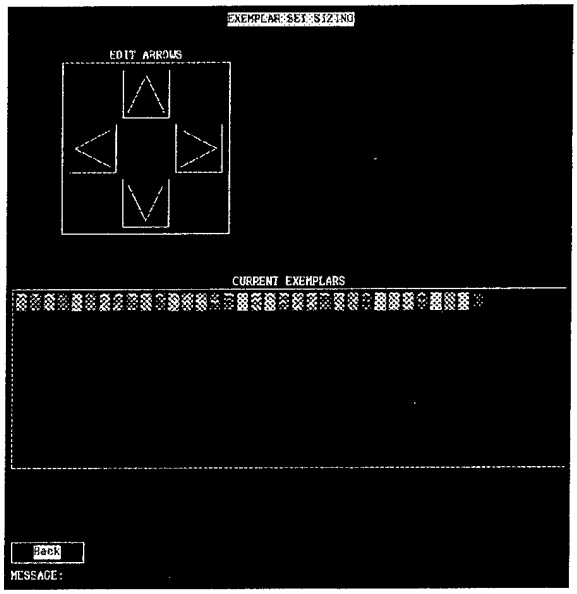


Figure A-7. Exemplar Set Sizing

A5.4 NET TRAINER

From the MAIN screen, select the NETTRAINER command to enter the NET TRAINER screen, seen in Figure A-8 below. A query to use the existing net configuration appears. Enter 'yes' to use the existing net or 'no' if you wish to change the hidden layer nodes value. If 'no' has been selected, input the number of hidden layer nodes, then answer 'yes' to the interaction logging query. If application of the existing net is selected, proceed to the SET ITER command. A default number of 10 iterations is selected. The range of possible iterations is 1 to 200. After the iteration value has been entered, execute the SETNOISE command. A default number of 64 for noise amplitude is displayed. The amplitude range is 0 to 240. After a noise value is entered, execute the BLOBS command if random patterns of noise corruption are required. A default number of 240 for noise amplitude is displayed. The range of noise amplitude is 0 to 255. After a blob value is entered, execute the BLOBS command. To modify either foreground or background luminance, execute either of the SETFORE/SETBACK commands. Use the keyboard to define the level of contrast for foreground/background. Execute the SHIFT command to define the degree of exemplar character shift within the character matrix. Use the keyboard to enter the integral number of pixels to shift the exemplar in either the horizontal or vertical direction. Upon completion of defining the corruption parameters, execute the TRAIN NET command to begin training the net. Upon activation of the net, the NET TRAINER screen displays the defined exemplar set in the left most column of the window. See figure A-TBD, TRAINER NET screen. A row by column matrix displays the bit map for each exemplar and the character identified by the trainer net. The top row of the screen displays the active method of corruption as it is applied to each row of exemplars.

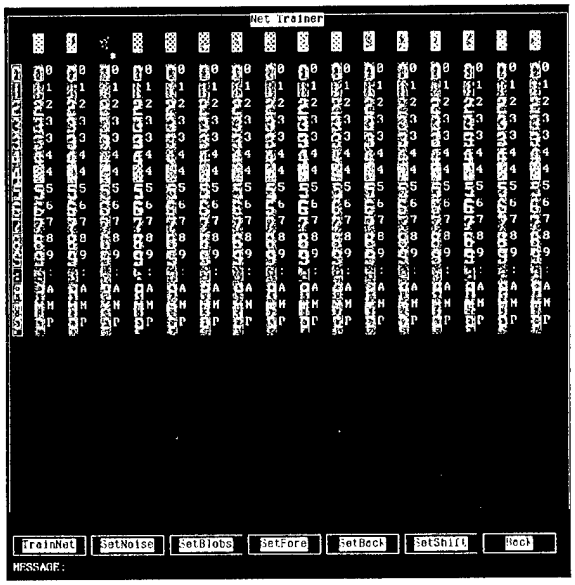


Figure A-8. Net Trainer

The NET TRAINER iterates through a back-propagation algorithm until the neural network output converges toward the exemplar set. Upon convergence, the final net coefficients are recorded. Neural net training results are reflected in confidence levels. Confidence levels are bounded by best and worst cases. Training results are displayed at the bottom of the NET TRAINER screen. Results are updated after each iteration. Results are displayed as follows:

COUNT	=	the number of iterations.
CONF	=	the level of confidence during training of the neural network.
WORST CONF	=	the worst level of confidence during training of the neural network.
BEST CONF	=	the best level of confidence during training of the neural network.
B COUNT	=	the best count.

Upon completion of training, save the net data.

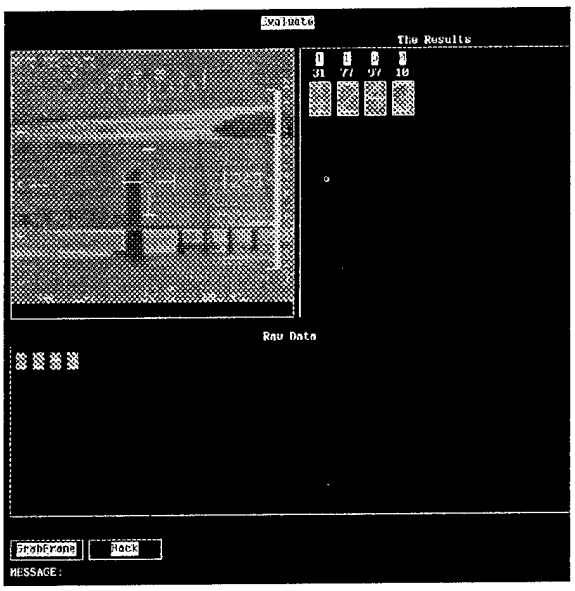


Figure A-9. Evaluate

A5.5 EVALUATE SCREEN

From the MAIN screen, select the EVAL NET command to enter the NET EVALUATE screen, seen in Figure A-9 above. The EVALUATE screen function provides the user with an interface to perform video character recognition using a trained net. The CURRENT FRAME window (upper left quadrant of the EVALUATE screen) displays the most recently captured video frame. The RESULTS window (upper right quadrant) displays three fields of interest for each character field being evaluated. The three fields include the net-identified character, the evaluated confidence (0 to 100 displayed below the identified character) and the bit map of the character. The RAW DATA window displays each raw character before sizing. Execute the GRAB FRAME command to capture another frame from the video source for evaluation.

APPENDIX B

DEVELOPERS GUIDE

The following paragraphs contain the process specifications for the VCRS system software design. The process specifications are related to the data flow diagrams located in section 4.1, Software Development. Note, each process specification is denoted with a mini-specification number MS/(number). The number represents the structural level of the subprocess. In addition, the programming design language for the Trainer process is included.

B.1 FIELD EDITOR PROCESS SPECIFICATION, MS/1.

```
%Subprogram Field Editor
.      (1)
.
%Inputs
   Field_Definition
   Mouse_Button   (Control)
   Mouse_Location
   Number_Characters
   Video_Frame
.
%Outputs
   Displays
   Field_Definition
   Grab_Frame   (Control)
.
.FUNCTION:
   Decode Mouse_Location and Mouse_Button
   Define Location of Fields in Video_Frame
   Define Character Locations in each Field
   Display Video_Frame, Field and Character  Locations
   Output(Displays, Field_Definition)
END
```

B.2 EXEMPLAR EDITOR PROCESS SPECIFICATION, MS/2.

```
%Subprogram Exemplar Editor
.      (2)
.
%Inputs
   Character_ID
   Character_Set
   Field_Definition
   Mouse_Button   (Control)
   Mouse_Location
   Video_Frame
.
%Outputs
   Character_Set
   Displays
```

```
   Grab_Frame   (Control)
.
.FUNCTION:
   Decode Mouse_Location and Mouse_Button
    Extract Character from       Video_Frame
   Read Character_ID
   Add Character and Character_ID to Character_Set
   Display Character_Set
   Output(Displays, Field_Definition)
```

B.3 SCALE CHARACTER PROCESS SPECIFICATION, MS/3.2.

```
%Subprogram Scale Character set
.      (3.2)
.
%Inputs
   Character_Set
   New_Character_Size
.
%Outputs
   Exemplar_Set
.
.FUNCTION:

   For Each Character in Character_Set
      Scale Character to New_Character_Size using Nearest Neighbor
      Store Character in Exemplar_Set
   End For
   Output(Exemplar_Set)
End
```

B.4 BACKPROP PROCESS SPECIFICATION, MS/4.10

```
%Subprogram Backprop
.      (4.10)
.
%Inputs
   Character_ID
   Neural_Net
.
%Outputs
   Neural_Net
.
.FUNCTION:

   Back Propagate using the Fix optimized back propagation
                  algorithm described in "Optimizing Back Propagation",
     Proceedings, AAAIC, 1988
```

B.5 CREATE NET TEXT PROCESS SPECIFICATION, MS/4.2.

```
%Subprogram Create Net
.       (4.2)
.
%Inputs
   Exemplar_Set
   Number_Hidden_Nodes
.
%Outputs
   Neural_Net
.
.FUNCTION:

   Compute Number of Input Nodes using Exemplar_Set.Character_Size
   Compute Number of Output Nodes using Exemplar_Set.Character_ID
   Create Neural Network using Number_Input_Nodes, Number_Hidden_Nodes, and
         Number_Output_Nodes
   Initialize Net
   Output(Neural_Net)
End
```

B.6 SHUFFLE SET TEXT PROCESS SPECIFICATION, MS/4.3.

```
%Subprogram Shuffle Set
.       (4.3)
.
%Inputs
   Exemplar_Set
.
%Outputs
   Raw_Character
.
.FUNCTION:

   Randomize Order of Characters in Exemplar_Set
   For Every Character in Exemplar_Set
      Output(Raw_Character)
   End For

End
```

B.7 CREATE STRUCTURE NOISE PROCESS SPECIFICATION, MS/4.4.

```
%Subprogram Create Structured Noise
.       (4.4)
.
%Inputs
   Structure_Parameters
```

```
.
%Outputs
   Noise_Mask
.
.FUNCTION:

   Generate Random Number of Vertices for Polygon
   Scan Convert Polygon into Noise_Mask.Pixels Using
      Structure_Parameters.Structure_Intensity
   Output(Noise_Mask)
End
```

B.8 FILTER PROCESS SPECIFICATION, MS/4.5.

```
%Subprogram Filter
.       (4.5)
.
%Inputs
   Raw_Character
.
%Outputs
   Enhanced_Character
.
.FUNCTION:

   Temp_Character = 3x3 Median Filter(Raw_Character)
   Enhanced_Character = Convolve Temp_Character with Mask
                        2  1  0
                        6      1 -6
                        0 -1 -2
   Output(Enhanced_Character)


End
```

B.9 TRANSLATE TEXT PROCESS SPECIFICATION, MS/4.6.

```
%Subprogram Translate
.       (4.6)
.
%Inputs
   Enhanced_Character
   Translation_Vector
.
%Outputs
   Shifted_Character
.
.FUNCTION:

   Compute Average of Enhanced_Character.Pixels
```

```
  Translate Enhanced_Character by Translation_Vector into Shifted_Character
  Fill Unused border areas in Shifted_Character with Average
  Output(Shifted_Character)
End
```

B.10 ADD STRUCTURED NOISE PROCESS SPECIFICATION, MS/4.7

```
%Subprogram Add Structured Noise
.      (4.7)
.
%Inputs
  Noise_Mask
  Shifted_Character
.
%Outputs
  Masked_Character
.
.FUNCTION:

  For Every Pixel in Shifted_Character
    If Shifted_Character.Pixel is less than Noise_Mask.Pixel then
      Masked_Character.Pixel = Noise_Mask.Pixel
    else
      Masked_Character.Pixel = Shifted_Character.Pixel
    End If
  End For
  Output(Masked_Character)
End
```

B.11 ADD RANDOM NOISE PROCESS SPECIFICATION, MS/4.8

```
%Subprogram Add Random Noise
.      (4.8)
.
%Inputs
  Masked_Character
  Noise_Amplitude
.
%Outputs
  Processed_Character
.
.FUNCTION:

  For Every Pixel in Masked_Character
    Compute Random Number between -Noise_Amplitude/2
        and Noise_Amplitude/2
    Processed_Character = Masked_Character + Random Number
  End For
  Output(Processed_Character)
End
```

B.12 RECOGNIZE TEXT PROCESS SPECIFICATION, MS/4.9

```
%Subprogram Recognize
.       (4.9)
.
%Inputs
   Neural_Net
   Processed_Character
.
%Outputs
   Confidence
   Net_Character_ID
   Neural_Net
.
.FUNCTION:

   Normalize Processed_Character.Pixels
   Compute Output_Weights using Normalized Pixels and Neural_Net
   Compute Net_Character_ID Using Output_Weights
   Compute Confidence Using Output_Weights
   Output(Confidence, Net_Character_ID)
End
```

B.13 RECOGNIZE PROCESS SPECIFICATION, MS/5.2

```
%Subprogram Recognize
.       (5.2)
.
%Inputs
   Character_Data
   Neural_Net
.
%Outputs
   Confidence
   Net_Character_ID
.
.FUNCTION:
   Normalize Character_Data.Pixels
   Compute Output_Weights using Normalized Pixels and Neural_Net
   Compute Net_Character_ID Using Output_Weights
   Compute Confidence Using Output_Weights
   Output(Confidence, Net_Character_ID)
End
```

B.14 EXTRACT CHARACTER PROCESS SPECIFICATION, MS/5.3

```
%Subprogram Extract Character
.       (5.3)
.
```

```
%Inputs
  Field_Definition
  Video_Frame
.
%Outputs
  Character_Data
.
.FUNCTION:

  For Each Character in Field_Definition
    Extract Character_Data from Video_Frame
    Output(Character_Data)
  End For
End
```

B.15 TRAINER PDL.

Trainer Data

Iterations: 0 < n <= 200

Corruption data:

Noise: on/off

1 <= max amplitude <= 240
all pixels are transformed
New Pixel := old pixel + random between 0 and max amplitude

Blobs: on/off

random(3-12) sided polygon
vertices are computed randomly

Shift: on/off

translates vertically -10 <= n <= 10 rows
translates horizontally -10 <= n <= 10 columns

---

trainer:

Training mode = false
while accumulated confidence < 0.85 or worst confidence < -0.50 loop

for number iterations loop

```
            Shuffle(randomize) raw training set

            if blobs enabled

                  create blob (polygon with random intensity)

            end if

            for each character in training set

                  shift vertical (random(-number pixels, number pixels))
                  shift horizontal (random(-number pixels, number pixels))

                  add random noise to all pixels in character


                  recognize character and get confidence

                  if net identified character wrong then
                        confidence = -confidence

                  add confidence to accumulated confidence

                  if in training mode then
                        run backprop function


            end character in training set loop

      end iteration loop

      accumulated confidence := accumulated confidence / number iterations

      toggle training mode

end while

-----------------------------------------------------------------
-----------------------------------------------------------------


add_Noise: all pixels transformed
            New Pixel := old pixel + random between 0 and noise level

            if new pixel < 4 then
                  new pixel := 0
            if new pixel > 255 then
                  new pixel := 255

blobs:
            random(3-12) sided polygon
            vertices are computed randomly
```

bitmap rescaling:

> During evaluation, the source bitmap will be scaled to the size of the training set bitmaps (set in the exemplar size function). No interpolation is being performed. Rows and columns are just being thrown away or replicated.

This page intentionally left blank.

APPENDIX C

SOURCE CODE

This section contains the following source code files developed for the VCRS application.

C.1 CONCAVE.C

```
/*
* Concave Polygon Scan Conversion
* by Paul Heckbert
* from "Graphics Gems", Academic Press, 1990
*/

/*
* concave: scan convert nvert-sided concave non-simple polygon with vertices
at
* (point[i].x, point[i].y) for i in [0..nvert-1] within the window win by
* calling spanproc for each visible span of pixels.
* Polygon can be clockwise or counterclockwise.
* Algorithm does uniform point sampling at pixel centers.
* Inside-outside test done by Jordan's rule: a point is considered inside if
* an emanating ray intersects the polygon an odd number of times.
* drawproc should fill in pixels from xl to xr inclusive on scanline y,
* e.g:
*     drawproc(y, xl, xr)
*     int y, xl, xr;
*     {
*         int x;
*         for (x=xl; x<=xr; x++)
*           pixel_write(x, y, pixelvalue);
*     }
*
*  Paul Heckbert  30 June 81, 18 Dec 89
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>


#include "concave.h"

#define ALLOC(ptr, type, n)  ASSERT(ptr = (type *)malloc((n)*sizeof(type)))


typedef struct {        /* a polygon edge */
   double x;      /* x coordinate of edge's intersection with current scanline
*/
   double dx;     /* change in x with respect to y */
   int i;   /* edge number: edge i goes from pt[i] to pt[i+1] */
} Edge;


typedef struct {                /* window: a discrete 2-D rectangle */
```

```
    int x0, y0;                 /* xmin and ymin */
    int x1, y1;                 /* xmax and ymax (inclusive) */
} Window;

static Window theWin;
static Window *win = &theWin;


static int n;                   /* number of vertices */
static Point2 *pt;              /* vertices */

static int nact;        /* number of active edges */
static Edge *active;            /* active edge list:edges crossing scanline y */

int compare_ind(), compare_active();

static void delete(int);            /* remove edge i from active list */
static void insert(int, int);       /* append edge i to end of active list */


static void delete(i)           /* remove edge i from active list */
int i;
{
    int j;

    for (j=0; j<nact && active[j].i!=i; j++);
    if (j>=nact) return; /* edge not in active list; happens at win->y0*/
    nact--;
/*   bcopy(&active[j+1], &active[j], (nact-j)*sizeof active[0]); */
    memcpy(&active[j], &active[j+1],  (nact-j)*sizeof active[0]);

}

static void insert(i, y)            /* append edge i to end of active list */
int i, y;
{
    int j;
    double dx;
    Point2 *p, *q;

    j = i<n-1 ? i+1 : 0;
    if (pt[i].y < pt[j].y) {p = &pt[i]; q = &pt[j];}
    else                {p = &pt[j]; q = &pt[i];}
    /* initialize x position at intersection of edge with scanline y */
    active[nact].dx = dx = (q->x-p->x)/(q->y-p->y);
    active[nact].x = dx*(y+.5-p->y)+p->x;
    active[nact].i = i;
    nact++;
}

/* comparison routines for qsort */
compare_ind(u, v) int *u, *v; {return pt[*u].y <= pt[*v].y ? -1 : 1;}
compare_active(u, v) Edge *u, *v; {return u->x <= v->x ? -1 : 1;}


void Concave(int nvert, Point2 *point,  void (*spanproc)())
{
```

```
int k, y0, y1, y, i, j, xl, xr;
    int *ind;        /* list of vertex indices, sorted by pt[ind[j]].y */


n = nvert;
pt = point;
if (n<=0) return;
ALLOC(ind, int, n);
ALLOC(active, Edge, n);

/* create y-sorted array of indices ind[k] into vertex list */
for (k=0; k<n; k++)
   ind[k] = k;
qsort(ind, n, sizeof ind[0], compare_ind);   /* sort ind by pt[ind[k]].y */

nact = 0;                          /* start with empty active list */
k = 0;                      /* ind[k] is next vertex to process */
y0 = MAX(win->y0, ceil(pt[ind[0]].y-.5));          /* ymin of polygon */
y1 = MIN(win->y1, floor(pt[ind[n-1]].y-.5)); /* ymax of polygon */

for (y=y0; y<=y1; y++) {          /* step through scanlines */
   /* scanline y is at y+.5 in continuous coordinates */

   /* check vertices between previous scanline and current one, if any */
   for (; k<n && pt[ind[k]].y<=y+.5; k++) {
       /* to simplify, if pt.y=y+.5, pretend it's above */
       /* invariant: y-.5 < pt[i].y <= y+.5 */
       i = ind[k];
       /*
        * insert or delete edges before and after vertex i (i-1 to i,
        * and i to i+1) from active list if they cross scanline y
        */
       j = i>0 ? i-1 : n-1;       /* vertex previous to i */
       if (pt[j].y <= y-.5)       /* old edge, remove from active list */
         delete(j);
       else if (pt[j].y > y+.5)  /* new edge, add to active list */
         insert(j, y);
       j = i<n-1 ? i+1 : 0;       /* vertex next after i */
       if (pt[j].y <= y-.5)       /* old edge, remove from active list */
         delete(i);
       else if (pt[j].y > y+.5)  /* new edge, add to active list */
         insert(i, y);
   }

   /* sort active edge list by active[j].x */
   qsort(active, nact, sizeof active[0], compare_active);

   /* draw horizontal segments for scanline y */
   for (j=0; j<nact; j+=2) {      /* draw horizontal segments */
       /* span 'tween j & j+1 is inside, span tween j+1 & j+2 is outside */
       xl = ceil(active[j].x-.5);             /* left end of span */
       if (xl<win->x0) xl = win->x0;
       xr = floor(active[j+1].x-.5);   /* right end of span */
       if (xr>win->x1) xr = win->x1;
       if (xl<=xr)
         (*spanproc)(y, xl, xr);       /* draw pixels in span */
          active[j].x += active[j].dx; /* increment edge coords */
       active[j+1].x += active[j+1].dx;
```

```
      }
       }

       free(active);
       free (ind);
}


void Setup_World (int MinX, int MinY, int MaxX, int MaxY)
{
   win->x0 = MinX;
   win->y0 = MinY;
   win->x1 = MaxX;
   win->y1 = MaxY;
}
```

C.2 BACKPROP.C

```
/* GENERALIZED BACKPROPAGATION ALGORITHM, TRAINED THRESHOLDS,
 ALL INPUTS AND OUTPUTS NORMALIZED 0 -- 1                                  */

 #include <stdio.h>
 #include <stdlib.h>
 #include <ctype.h>
 #include <conio.h>

 #include "net.h"
 #include "network.h"

 #define out_error(k,out) (((k)==(out))?1.0f:0.0f)

 int I, J, K, trwidth, trheight;
 char  * inchar;
 float * input;
 float * hidden;
 float * output;
 float * w1;
 float * w2;
 float * thetaj;
 float * thetak;

 static float ETA       = 0.8f;
 static float ALPHA     = 0.4f;
 static float *sum, *w1p, *w2p, *w1_save, *w2_save;
 static float *thetaj_save, *thetak_save;

 void randomize_weights ( void );
 float fabs1 ( float );
 float sign ( float );
 float u ( float );
 void print_float_matrix ( FILE * , float * , int );


 int init_backprop ( void )
      {
```

```
      int v, num_inchars=0, gotit, gotnet=0, i, k;
      float *fptr1, *fptr2;
      char str[95];

      /* Go get the training data and create the network */
      if ( get_train_set ( filtrainset ) ) return -1;
      i = I;
      k = K;

      /* See if there is a network in the directory.  If so, ask user
            if he wants to use it for the training */
      if ( ! read_weights() )
            {
            gotnet = 0;

            /* See if number of inputs and outputs = training set */
            if ( ( i == I ) && ( k == K ) )
                  {
                  if ( confirm ( "Do you want to use the existing net? " ) > 0
)
                        gotnet = 1;
                  else
                        kill_net();
                  }
            else kill_net();
            }

      if ( NOT gotnet )
            {
            /* Take this path for a new network */
            /* We need get the number of hidden layer nodes and allocate the
net */
            I = i;
            K = k;
            /* User defines the number of hidden layer nodes */
            J  = get_number_from_user("Input the number of hidden layer nodes:
");
            clear_message ( );
            if ( make_net () )
                  {
                  put_message_with_wait ( "make_net (malloc) error !!\0" );
                  return -1;
                  }
            }



      /* Now allocate memory for trainer specific stuff */
      w1p   = (float * ) (malloc(I*J*(sizeof(float))));
      w2p    = (float * ) (malloc(J*K*(sizeof(float))));
      sum = malloc ( J * sizeof ( float ) );
      if ( ( w1p == NULL ) || ( w2p == NULL ) || ( sum == NULL ) )
            {
            put_message_with_wait ( " error on allocating trainer specific
data\0" );
            return -1;
            }
```

```
      /* For an existing net, we need to assign indices to each training
            character according to its position in the output vector */
      if ( gotnet )
            {
            for ( i = 0; i < ntrain; i++ )
                  {
                  gotit = 0;
                  for ( k = 0; k < K; k++ )
                        {
                        if ( trainset[i].ch == inchar[k] )
                              {
                              gotit = 1;
                              trainset[i].out_index = k;
                              }
                        }
                  if ( NOT gotit )
                        {
                        put_message ( "Exemplar set contains different
characters than existing net\0" );
                        return -1;
                        }
                  }
            /* Now initialize w1p and w2p arrays */
            fptr1 = w1;
            fptr2 = w1p;
            for ( i=0; i<I*J; i++ )
                  *fptr2++ = *fptr1++;
            fptr1 = w2;
            fptr2 = w2p;
            for ( i=0; i<J*K; i++ )
                  *fptr2++ = *fptr1++;
            }
      else
            {
            /* This path will be taken for new nets */
            randomize_weights ();

            /* Set up the inchar array.  This contains the actual characters
                  aligned with the output array */
            for ( v=0; v<ntrain; v++ )
                  {
                  if ( v == 0 )  /* This gets the first character */
                        {
                        num_inchars = 1;
                        inchar[0] = trainset[v].ch;
                        trainset[v].out_index = 0;
                        }
                  else
                        {
                        /* Sort the training characters in ASCII sequence */
                        gotit = 0;
                        for ( i = 0; i<num_inchars; i++ )
                              {
                              if ( trainset[v].ch == inchar[i] )
                                    {
                                    trainset[v].out_index = i;
                                    gotit = 1;
                                    }
```

```
                        }
                    if ( ! gotit )
                        {
                        trainset[v].out_index = num_inchars;
                        inchar[num_inchars] = trainset[v].ch;
                        num_inchars ++;
                        }
                    }
                }

            /* Verify we processed the same number of outputs as defined in
                the training set file */
            if ( num_inchars != K )
                {
                sprintf ( str, "Num of train characters ( %d ) diff from num
of outputs ( %d ).  Resetting K !\0", num_inchars, K);
                put_message_with_wait ( str );
                K = num_inchars;
                clear_message ();
                }
            }

        /* Now allocate memory for the holding buffer for the best net */
        w1_save     = (float * ) (malloc(I*J*(sizeof(float))));
        w2_save     = (float * ) (malloc(J*K*(sizeof(float))));
        thetak_save = (float * ) (malloc(K*(sizeof(float))));
        thetaj_save = (float * ) (malloc(J*(sizeof(float))));
        if ( ( thetaj_save == NULL ) || ( w1_save == NULL ) ||
                ( thetak_save == NULL ) || ( w2_save == NULL ) )
                {
                put_message_with_wait ( "Error Allocating Best Net
Buffers\0" );
                return -1;
                }

        return 0;
        }


 void backprop ( int out )
        /*
         *  This function modifies the network based on the current input
         *  matrix (input) and the expected value of the output, which is
pointed
         *  to by the incoming index (out).  The net must have been run
         *  against the input array prior to coming into this routine.  This
         *  should be done by calling recognize with a dot matrix.
         */
        {
        float delta, ftemp, flip;
        int i, j, k, index;

        /* initialize for the backprop loop */
        for ( j=0; j<J; ++j ) sum[j] = 0.0f;

        /* Iterate the w2 and thetak arrays of the net */
        for(k=0; k<K; ++k)
                {
```

```
            /*  The next two lines implement the Fix optimized back
propagation
            algorithm described in "Optimizing Back Propagation", Proceedings,
            AAAIC, 1988
            */
            flip = sign ( out_error(k,out) - 0.5 ) * ( output[k] - 0.5 );
            delta = ( output[k] * (1 - output[k]) * u(flip) + 0.25 * u(-flip))
            * (out_error(k,out) - output[k]);

            for(j=0; j<J; ++j)
                  {
                  index = k*J + j;
                  ftemp = w2[index] + ETA * delta * hidden[j];
                  w2[index] = ftemp + ALPHA * ( ftemp - w2p[index] );
                  w2p[index] = w2[index];
                  sum[j] = sum[j] + delta * w2[index];
                  }
            thetak[k] = thetak[k] - ETA * delta;
            }

      /* Now iterate the w1 and thetaj arrays of the net */
      for(j=0; j<J; ++j)
            {
            delta = hidden[j] * (1.0f - hidden[j]) * sum[j];
            for(i=0; i<I; ++i)
                  {
                  index = j * I + i;
                  w1[index] = w1[index] + ETA * delta * (input[i] - 0.5) +
                              ALPHA * (w1[index] - w1p[index]);
                  w1p[index] = w1[index];
                  }
            thetaj[j] = thetaj[j] - ETA * delta;
            }
      }


float net_cost ( int out )
      /* This function runs the net and computes an accumlated cost.
            It does not train the net, and is intended to verify that
            the net really is better */
      {
      float cost=0.0f;
      int k;

      /* run the net on the character currently in the input vector */
      net();

 /* now compute the acumulated cost of each of the outputs */
 for ( k=0; k<K; ++k )
     cost = cost + fabs1 ( output[k] - out_error(k,out) );
 return(cost);
 }


float fabs1 ( float x )
 {
 if ( x < 0.0f )
    return ( -x );
```

```
 else
    return ( x );
 }


float sign ( float x)
     {
     if(x < 0.0f )
           return ( -1.0f );
     else
           return ( 1.0f );
     }


float u ( float x )
     {
     if(x < 0.0f )
           return ( 0.0f );
     else
           return ( 1.0f );
     }


void randomize_weights ( void )
 {
 int i, j, k, index;

 /* initialize the random number generator */
 srand(5000);

 /* initialize the thetaj vector */
 for(j=0; j<J; ++j)
         thetaj[j] = ( (float) rand() ) / 32768.0f - 0.5f;

 /* now initialize the thetak vector */
 for(k=0; k<K; ++k)
     thetak[k] = ( (float) rand() ) / 40000.0f - 0.5f;

 /* now initialize the w1 matrix */
 for(j=0; j<J; ++j)
     {
         for(i=0; i<I; ++i)
             {
         index = j*I+i;
           w1[index] = ( (float) rand() ) / 40000.0f - 0.5f;
             w1p[index] = w1[index];
         }
         }

 /* now initialize the w2 matrix*/
 for(k=0; k<K; ++k)
           {
     for(j=0; j<J; ++j)
           {
           index = k*J + j;
           w2[index] = ( (float) rand() ) / 40000.0f - 0.5f;
           w2p[index] = w2[index];
           }
```

```
    }
 }


void kill_backprop ( void )
 {

     free ( w1p );
 free ( w2p );
 free ( sum );
 free ( w1_save );
 free ( w2_save );
 free ( thetaj_save );
 free ( thetak_save );
 }


int save_weights ( void )
 {
 int k;
 FILE * wts;

     wts = fopen( filnet, "w");
     if(wts == 0)
            {
            put_message_with_wait ( "File Access Error Saving Weights\n" );
            return -1;
           }

     fprintf ( wts,"%d  %d  %d  %d  %d\n", I, J, K, trwidth, trheight );
     for ( k=0; k<K; k++ )
            fprintf ( wts,"   %x", (unsigned int)inchar[k] );
     fprintf ( wts, "\n" );

     print_float_matrix ( wts, thetaj_save, J );
     print_float_matrix ( wts, thetak_save, K );
     print_float_matrix ( wts, w1_save, I*J );
     print_float_matrix ( wts, w2_save, J*K );

     fprintf(wts, "\n");
 fclose(wts);
 return 0;
 }


void print_float_matrix ( FILE * fil, float * mat, int len )
 {
 int count;

 for ( count = 0; count < len; count ++ )
     {
     if ( ( count % 5 ) == 0 )
                     fprintf ( fil, "\n" );
             fprintf ( fil, " %12f ", mat[count] );
     }
      fprintf ( fil, "\n" );

      return;
```

```
}


void get_weights ( void )
      {
      float *src, *dest;
      int i;

      /* copy w1 */
      src = w1_save;
      dest = w1;
      for ( i = 0; i<(I*J); i++ )
            *dest++ = *src++;

      /* copy w2 */
      src = w2_save;
      dest = w2;
      for ( i = 0; i<(J*K); i++ )
            *dest++ = *src++;

      /* copy thetaj */
      src = thetaj_save;
      dest = thetaj;
      for ( i = 0; i<J; i++ )
            *dest++ = *src++;

      /* copy thetak */
      src = thetak_save;
      dest = thetak;
 for ( i = 0; i<K; i++ )
     *dest++ = *src++;

 }


void hold_weights ( void )
 {
 float *src, *dest;
 int i;

 /* copy w1 */
 src = w1;
 dest = w1_save;
 for ( i = 0; i<(I*J); i++ )
     *dest++ = *src++;

 /* copy w2 */
 src = w2;
 dest = w2_save;
 for ( i = 0; i<(J*K); i++ )
     *dest++ = *src++;

 /* copy thetaj */
 src = thetaj;
 dest = thetaj_save;
 for ( i = 0; i<J; i++ )
    *dest++ = *src++;
```

```
  /* copy thetak */
  src = thetak;
  dest = thetak_save;
  for ( i = 0; i<K; i++ )
      *dest++ = *src++;

  }
```

EVAL.C

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <float.h>
#include <conio.h>    /* temporary for demo */
#include <dos.h>

#include "net.h"
#include "network.h"
#include "ntiga.h"
#include "vcrsmfg.h"

/* Function prototypes */
void pre_filter (unsigned char *, unsigned int , unsigned int );

static void convert_chars ( void );
int grab_eval_frame ( void );

static command_type grab      =
      {     "GrabFrame",
            grab_eval_frame,
            "Grab a frame and redraw fields" };



void save_bitmap(int num, unsigned char *bitmap)
{
      FILE *fp;
      char fname[40];


      sprintf(fname,"bitmap%d.raw\0",num);
      fp = fopen(fname,"w");

      fwrite (bitmap, trwidth*trheight, 1, fp);

      fclose (fp);
}



/*********************************************************/
/*                                                       */
/*                                                       */
/*                                                       */
/*                                                       */
/*********************************************************/
```

```
int eval_net( void )
        {
        box_type eval_box;
        box_type data_box;

        /* Initialize display and data setup */
        clear_display();

        put_screen_title ( "Evaluate" );

        eval_box.pt1.x = frame_box.pt2.x + MARGIN;
        eval_box.pt1.y = frame_box.pt1.y;
        eval_box.pt2.x = display_size.x - 2;
        eval_box.pt2.y = frame_box.pt2.y  ;

        put_box_title ( "The Results", &eval_box );

        data_box.pt1.x = frame_box.pt1.x ;
        data_box.pt1.y = frame_box.pt2.y + 50;
        data_box.pt2.x = display_size.x - 2;
        data_box.pt2.y = display_size.y - 125 ;

        put_box_title ( "Raw Data", &data_box );


        if ( get_field_set() || read_weights() )
                {
                put_message_with_wait ( "Error during evaluator setup\0" );
                return -1;
                }

        /* This function does all the work */
        convert_chars();

        /* We're closing down, so kill evaluator */
        kill_field_set ();
        kill_net();
        clear_display();

        return ( 0 );
        }


static void convert_chars( void )
       {
       short i, j, x, y, xstart, ystart, xinc, yinc, cw, ch, iconf;
       char temp_char[2] = { 'a', '\0' };
       char buf[10];
       unsigned char * in;
       box_type dmbox, destbox, copybox;
       point_type dispPt;
       float conf;

       float xfact, yfact;
       BYTE *newdm;
       int k,m, ox, oy;

       int rep=0;
```

```
      /*
       *  copybox is where all dot matrices will be put prior to bringing
       *  the data into memory.  It has the dimensions of the training set.
       *  Characters which are not the correct size will be zoom-BLTed into
       *  copybox, while others are simply copied.
       */


      xstart = frame_box.pt2.x + 2*MARGIN + 5;
      ystart = frame_box.pt1.y + MARGIN + 2*font.charhigh;
      xinc = mag * trwidth + MARGIN;
      if ( xinc < 2*font.charwide + MARGIN ) xinc = 2 * font.charwide +
MARGIN;
      yinc = mag * trheight + 2 * font.charhigh + MARGIN;

      get_frame( );     /* and get a frame */
      draw_box ( &frame_box );
      clear_message ( );

      do
            {
            /* Initialize for this pass */


            copybox.pt1.x = 10 + MARGIN;
            copybox.pt1.y = frame_box.pt2.y + 60;
            copybox.pt2.x = copybox.pt1.x + trwidth - 1;
            copybox.pt2.y = copybox.pt1.y + trheight - 1;


            enable_command ( &grab );

            _fpreset();                  /* Reset floating point processor */

            x = xstart;                  /* For display of data only */
            y = ystart;

            /* Now loop thru each field */
            for (i = 0; i < nfields; i++)
                  {
                  cw = fieldset[i].cwidth;
                  ch = fieldset[i].cheight;

                  /* And loop thru each character in the field */
                  for (j = 0; j < fieldset[i].nchars; j++)
                        {

                        /* Set up for the characters' source box */
                        dmbox.pt1.x = frame_box.pt1.x +
fieldset[i].points[j].x;
                        dmbox.pt1.y = frame_box.pt1.y +
fieldset[i].points[j].y;
                        dmbox.pt2.x = dmbox.pt1.x + cw - 1;
                        dmbox.pt2.y = dmbox.pt1.y + ch - 1;
```

```
                        /*  Now get the character dot matrix */
                        in = get_dm ( &dmbox );
                        if ( in == NULL )
                              {
                              put_message_with_wait ( "get_dm in eval failed.
!" );
                              return ;
                              }


                        /* scale the sucker if needed */
                        if ( ( cw != trwidth ) || ( ch != trheight ) )
                        {
                              xfact = ((float)trwidth / (float)cw);
                              yfact = ((float)trheight / (float)ch);

                              newdm = ( BYTE * ) malloc ( trheight * trwidth
);
                              for ( k=0; k < trheight; k++ )
                              {
                                    oy = (int) ((float)k) / yfact;
                                    for ( m=0; m < trwidth; m++ )
                                    {
                                          ox = (int) ((float)m) / xfact;
                                          newdm [ k * trwidth + m ] = in
[oy*cw+ox];
                                    }
                              }
                              free (in);
                              in = newdm;
                        }
                        pre_filter (in, trwidth, trheight);

                        put_dm (in, &copybox);


/*                      if (j == 3)
                        {
                              save_bitmap(rep,in);
                              rep++;
                        }*/
/*                      for (rep=0;rep < 6;rep++)
                        {*/

                        /* Submit the character to the net */
                        temp_char[0] = recognize( in, &conf );

/*                      printf("%f  ", conf);
                        }
                        printf("\n"); */

                        /* And free the memory for the dot matrix */

                        free (in);

                        /* The following is just for display in evaluator */
```

```
                        /* Set up the box where we will put the magnified dot
matrix */
                        if ( x + xinc > display_box.pt2.x )
                              {
                              x = xstart + xinc;
                              y += yinc;
                              }
                        destbox.pt1.x = x;
                        destbox.pt1.y = y;
                        destbox.pt2.x = destbox.pt1.x + mag * trwidth - 1;
                        destbox.pt2.y = destbox.pt1.y + mag * trheight - 1;

                        /* Set up the point where we will put the nets' return
*/
                        disppt.x = destbox.pt1.x/2 + destbox.pt2.x/2 -
font.charwide/2;
                        disppt.y = destbox.pt1.y - 5 -2*font.charhigh;

                        /* Now, zoom the dot matrix, and put the character
below it. */


                        zoom_box ( &copybox, &destbox );

/*                      put_dm(in, &destbox); */

                        out_text ( disppt, temp_char, 1, REVERSE_TEXT );

                        /* Set up the point where we will put the nets'
confidence */
                        disppt.x = destbox.pt1.x/2 + destbox.pt2.x/2 -
2*font.charwide/2;
                        disppt.y = destbox.pt1.y - 2 - font.charhigh;
                        if ( conf >= 1.0f ) buf[0] = buf[i] = '*';
                        else
                              {
                              iconf = conf*100;
                              sprintf ( buf, "%2.2d", iconf );
                              }
                        out_text ( disppt, buf, 2, NORMAL_TEXT );
                        x += xinc;

                        draw_box ( &destbox );

                        copybox.pt1.x += trwidth+MARGIN;
                        copybox.pt2.x += trwidth+MARGIN;

                        }   /* End j-character loop */

                  y += yinc;
                  x = xstart;

                  copybox.pt1.x = 10 + MARGIN;
                  copybox.pt1.y += trheight + MARGIN ;
                  copybox.pt2.x = copybox.pt1.x + trwidth - 1;
                  copybox.pt2.y = copybox.pt1.y + trheight - 1;

                  }   /* End field loop */
```

```
            } while ( NOT wait_for_user_event () );

      }
```

```
int grab_eval_frame ( void )
      {
      /* Tell the user */
      put_message ( "Grabbing a frame" );

      get_frame ( );

      clear_message();

      draw_box ( &frame_box );

      /* Fini */
      return 0;
      }
```

EVALTOM

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <float.h>
#include <conio.h>   /* temporary for demo */
#include <dos.h>

#include "net.h"
#include "network.h"
#include "ntiga.h"
#include "vcrsmfg.h"

/* Function prototypes */
static void convert_chars ( void );
int grab_eval_frame ( void );

static command_type grab      =
      {     "GrabFrame",
            grab_eval_frame,
            "Grab a frame and redraw fields" };


/*********************************************************/
/*                                                       */
/*                                                       */
/*                                                       */
/*                                                       */
/*********************************************************/
```

```
int eval_net( void )
        {
        box_type eval_box;
        box_type data_box;

        /* Initialize display and data setup */
        clear_display();

        put_screen_title ( "Evaluate" );

        eval_box.pt1.x = frame_box.pt2.x + MARGIN;
        eval_box.pt1.y = frame_box.pt1.y;
        eval_box.pt2.x = display_size.x - 2;
        eval_box.pt2.y = frame_box.pt2.y  ;

        put_box_title ( "The Results", &eval_box );

        data_box.pt1.x = frame_box.pt1.x ;
        data_box.pt1.y = frame_box.pt2.y + 50;
        data_box.pt2.x = display_size.x - 2;
        data_box.pt2.y = display_size.y - 125 ;

        put_box_title ( "Raw Data", &data_box );


        if ( get_field_set() || read_weights() )
                {
                put_message_with_wait ( "Error during evaluator setup\0" );
                return -1;
                }

        /* This function does all the work */
        convert_chars();

        /* We're closing down, so kill evaluator */
        kill_field_set ();
        kill_net();
        clear_display();

        return ( 0 );
        }


static void convert_chars( void )
       {
       short i, j, x, y, xstart, ystart, xinc, yinc, cw, ch, iconf;
       char temp_char[2] = { 'a', '\0' };
       char buf[10];
       unsigned char * in;
       box_type dmbox, destbox, copybox;
       point_type disppt;
       float conf;


       /*
        *  copybox is where all dot matrices will be put prior to bringing
        *  the data into memory.  It has the dimensions of the training set.
        *  Characters which are not the correct size will be zoom-BLTed into
```

```
         *  copybox, while others are simply copied.
         */


        xstart = frame_box.pt2.x + 2*MARGIN + 5;
        ystart = frame_box.pt1.y + MARGIN + 2*font.charhigh;
        xinc = mag * trwidth + MARGIN;
        if ( xinc < 2*font.charwide + MARGIN ) xinc = 2 * font.charwide +
MARGIN;
        yinc = mag * trheight + 2 * font.charhigh + MARGIN;

        get_frame( );      /* and get a frame */
        draw_box ( &frame_box );
        clear_message ( );

        do
              {
              /* Initialize for this pass */


              copybox.pt1.x = 10 + MARGIN;
              copybox.pt1.y = frame_box.pt2.y + 60;
              copybox.pt2.x = copybox.pt1.x + trwidth - 1;
              copybox.pt2.y = copybox.pt1.y + trheight - 1;


              enable_command ( &grab );

              _fpreset();                    /* Reset floating point processor */

              x = xstart;                    /* For display of data only */
              y = ystart;

              /* Now loop thru each field */
              for (i = 0; i < nfields; i++)
                    {
                    cw = fieldset[i].cwidth;
                    ch = fieldset[i].cheight;

                    /* And loop thru each character in the field */
                    for (j = 0; j < fieldset[i].nchars; j++)
                          {

                          /* Set up for the characters' source box */
                          dmbox.pt1.x = frame_box.pt1.x +
fieldset[i].points[j].x;
                          dmbox.pt1.y = frame_box.pt1.y +
fieldset[i].points[j].y;
                          dmbox.pt2.x = dmbox.pt1.x + cw - 1;
                          dmbox.pt2.y = dmbox.pt1.y + ch - 1;

                          /*
                           *  If the character is the correct size, copy it to
                           *  the region we do all retrievals from.  Otherwise,
the
                           *  box must be zoomed (or shrunk).
                           */
```

```
                        clear_box ( &copybox );

                        delay ( 5 );
                        if ( ( cw == trwidth ) && ( ch == trheight ) )
                              copy_box ( &dmbox, &copybox );
                        else
                              scale_box ( &dmbox, &copybox );
                        delay ( 5 );

                        /*  Now get the character dot matrix */
                        in = get_dm ( &copybox );

                        if ( in == NULL )
                              {
                              put_message_with_wait ( "get_dm in eval failed.
!" );
                              return ;
                              }

                        /* Submit the character to the net */
                        temp_char[0] = recognize( in, &conf );
                        /* And free the memory for the dot matrix */
                        free (in);

                        /* The following is just for display in evaluator */
                        /* Set up the box where we will put the magnified dot
matrix */
                        if ( x + xinc > display_box.pt2.x )
                              {
                              x = xstart + xinc;
                              y += yinc;
                              }
                        destbox.pt1.x = x;
                        destbox.pt1.y = y;
                        destbox.pt2.x = destbox.pt1.x + mag * trwidth - 1;
                        destbox.pt2.y = destbox.pt1.y + mag * trheight - 1;

                        /* Set up the point where we will put the nets' return
*/
                        disppt.x = destbox.pt1.x/2 + destbox.pt2.x/2 -
font.charwide/2;
                        disppt.y = destbox.pt1.y - 5 -2*font.charhigh;
                        /* Now, zoom the dot matrix, and put the character
below it. */
                        zoom_box ( &copybox, &destbox );
                        out_text ( disppt, temp_char, 1, REVERSE_TEXT );

                        /* Set up the point where we will put the nets'
confidence */
                        disppt.x = destbox.pt1.x/2 + destbox.pt2.x/2 -
2*font.charwide/2;
                        disppt.y = destbox.pt1.y - 2 - font.charhigh;
                        if ( conf >= 1.0f ) buf[0] = buf[i] = '*';
                        else
                              {
                              iconf = conf*100;
                              sprintf ( buf, "%2.2d", iconf );
                              }
```

```
                        out_text ( disppt, buf, 2, NORMAL_TEXT );
                        x += xinc;

                        draw_box ( &copybox );
                        draw_box ( &destbox );

                        copybox.pt1.x += trwidth+MARGIN;
                        copybox.pt2.x += trwidth+MARGIN;

                        }   /* End j-character loop */

                    y += yinc;
                    x = xstart;

                    copybox.pt1.x = 10 + MARGIN;
                    copybox.pt1.y += trheight + MARGIN ;
                    copybox.pt2.x = copybox.pt1.x + trwidth - 1;
                    copybox.pt2.y = copybox.pt1.y + trheight - 1;

                    }   /* End field loop */

                } while ( NOT wait_for_user_event () );

        }
```

```
int grab_eval_frame ( void )
        {
        /* Tell the user */
        put_message ( "Grabbing a frame" );

        get_frame ( );

        clear_message();

        draw_box ( &frame_box );

        /* Fini */
        return 0;
        }
```

C.5 EXEMPED

```
#include <ctype.h>
#include <conio.h>
#include <stdlib.h>
#include <malloc.h>
#include <dos.h>
#include <mem.h>

#include "net.h"
#include "mouse.h"
#include "vcrsmfg.h"
```

```
#include "ntiga.h"


/* Declare Globals */
int nexemp;                     /* # exemplar chars defined */
int nexemp_allocated;           /* # exemplar chars allocated */
exemp_type * exempset;          /* pointer to exemplar set array */

/* Declare Local variables */
static int exselect=-1;         /* Currently selected ex character */
static int old_exselect = -1; /* past value of exselect */
static box_type exemp_box;          /* Box where exemplar set will be drawn */

static int exemp_mag ;          /* exemplar editor mag factor */
static box_type editbox;        /* exemplar editor box */
static box_type orgbox;         /* raw exemplar editor box */
static BYTE *editbuffer;        /* exempler editor dm */
static BYTE fillcolor = 0;

/* Declare local functions */
static int  init_exemplar_editor ( void );
static void       exit_exemplar_editor ( void );
static int  grab_frame ( void );
static int  new_exemplar ( void );
static int  delete_exemplar ( void );
static int  change_exemp_id ( void );
static int  edit_exset ( void );
static void       draw_exemplar ( int );
static void       undraw_exemplar ( int );
static void       draw_exemplar_id_screen ( void );
static int  set_exemplar_data ( exemp_type * );
static int        add_exemplar ( exemp_type * );
static void       copy_exemplar ( exemp_type *, exemp_type * );
static int  isexemplar_selected ( void );
static void       unselect_exemplar ( void );
static int     magnify_exemplar (void );
static int     demagnify_exemplar (void );
static int  toggle_exemplar_color ( void );
static int  increase_threshold ( void );
static int  decrease_threshold ( void );
static void       fix_exemplar_box ( int );
static void    display_exemplar ( int );

static command_type grab      =
      {     "GrabFrame",
            grab_frame,
            "Grab a frame and redraw fields" };

static command_type new       =
      {     "NewExemp",
            new_exemplar,
            "Add character to the exemplar set" };

static command_type delete    =
      {     "DeleteEx",
            delete_exemplar,
            "Delete an exemplar character" };
```

```
static command_type changeid  =
      {     "ChangeID",
            change_exemp_id,
            "Change the ASCII character associated with an exemplar dot
matrix" };

static command_type editex    =
      {     "EditEx",
            edit_exset,
            "Edit the dot matrices and create a training set" };

static command_type magex =
      { "Magnify",
        magnify_exemplar,
        "Magnify the dot matrix" };

static command_type demagex =
      { "DeMagnify",
        demagnify_exemplar,
        "DeMagnify the dot matrix" };

static command_type fillcol =
      { "ToggleColor",
        toggle_exemplar_color,
        "Toggle fill color" };

static command_type incthresh =
      { "IncThresh",
        increase_threshold,
        "Increase contrast threshold" };

static command_type decthresh =
      { "DecThresh",
        decrease_threshold,
        "Decrease contrast threshold" };



int exemplar_editor ( void )
      /* This is the state machine for identifying the exemplar set */
      {

      if ( init_exemplar_editor() ) return -1;
      draw_exemplar_id_screen ();

      /* Command Processing Loop */
      do
            {
            /* Grab command is always enabled */
            enable_command ( &grab );

            /* If number of fields > 0, allow new exemplar */
            if ( nfields > 0 )
                  enable_command ( &new );

            /* If number of exemplars > 0, enable mods */
            if ( nexemp > 0 )
                  {
```

```
                enable_command ( &delete );
                enable_command ( &changeid );
                enable_command ( &editex );
                }

        clear_message ();

        } while ( NOT wait_for_user_event() );

    /* Save the data */
    exit_exemplar_editor ();

    /* Fini */
    return 0;
    }


int init_exemplar_editor ( void )
    {

    /* Initialize the fields display box */
    field_box.pt1.x = frame_box.pt2.x + MARGIN;
    field_box.pt2.x = display_box.pt2.x;
    field_box.pt1.y = frame_box.pt1.y;
    field_box.pt2.y = display_box.pt2.y - MARGIN;


    /* Go get the field data, and set up field boxes */
    if ( get_field_set() )
            /* Field data is bad.  Major stake in the heart */
        return -1;
    /* Fields must be ok.  Fix and draw them */
    fix_field_boxes ( -1, 0 );

    /* Since fields will not change, truncate the field box lower y */
    field_box.pt2.y = fieldset[nfields-1].fdbox.pt2.y + MARGIN;

    /* Now set up the exemplar set box. Do x'es */
    exemp_box.pt1.x = frame_box.pt1.x;
    /* If fields aren't using lower display, fill it with exemplar box */
    if ( field_box.pt2.y < frame_box.pt2.y )
        exemp_box.pt2.x = display_box.pt2.x;
    else
        exemp_box.pt2.x = frame_box.pt2.x;

    /* Now do y's */
    exemp_box.pt1.y = frame_box.pt2.y + MARGIN + font.charhigh + 1;
    exemp_box.pt2.y = display_box.pt2.y;

    /* Now get existing exemplar data, if it exists */
    if ( get_exemplar_set ( filexemp ) ) return -1;
    fix_exemplar_box ( -1 );

    /* Fini */
    return 0;
    }
```

```
void exit_exemplar_editor ( void )
      {

      /* See if we need to save the raw and normalized exemplar data */
      if ( ( ! fexist ( filexemp ) ) || ( fexist ( filexemp ) &&
                                    ( confirm ( "Do you want to save exemplar
data?" ) > 0 ) ) )
            save_exemplar_set ( filexemp );

      /* Now destroy dynamically allocated stuff */
      kill_exemplar_set ( );
      kill_field_set();

      /* Clear the display for the main application */
      clear_display ();

      /* Fini */
      }


int grab_frame ( void )
      {
      /* Tell the user */
      put_message ( "Grabbing a frame" );

      /* Now get the frame and draw fields */
      get_frame ( );
      draw_field ( -1 );
      clear_message();

      draw_box ( &frame_box );

      /* Fini */
      return 0;
      }



void draw_exemplar_id_screen ( void )
      {
      /* Draw the field editor screen */
      put_screen_title ( "EXEMPLAR SET EDITOR" );
      put_box_title ( "CURRENT FRAME", &frame_box );
      put_box_title ( "CURRENT FIELDS", &field_box );

      /* Grab a frame and draw fields */
      grab_frame ();

      put_box_title ( "CURRENT EXEMPLARS", &exemp_box );

      /* Fix the exemplar boxes and draw */
      draw_exemplar (-1);
      /* Fini */
      }
```

```
int new_exemplar ( void )
/*
 *    This function is invoked when the user wants to define a new
 *    exemplar character.  The user must place the mouse in one of the
 *    field characters and hit the mouse button.  The user must identify
 *    the character.  Then and only then, is the character dot matrix
 *    fetched, and the exemplar array updated.  Return is always zero.
 */
      {
      int itemp;
      exemp_type ex;

      /* Wait for user to identify a character in the field box */
      do    {

            /* Enable frame grab command */
            enable_command ( &grab );

            /* See if user is in a field character box */
            if ( isfieldchar_selected () )
                  {
                  /* Ask what it is */
                  put_message ( "Use keyboard to identify the character: " );
                  itemp = wait_for_buffered_kbin ( 1 );
                  /* Allow an abort */
                        if ( itemp < 0 ) continue;

                  /* Verify it's a printable char */
                  if ( isprint ( kbin_buf[0] ) )
                        {
                        /* Good character.  Set up the exemplar */
                        ex.ch = kbin_buf[0];
                        if ( set_exemplar_data ( &ex ) ) return -1;

                        /* and add to the array */
                        if ( add_exemplar ( &ex ) ) return -1;

                        }
                  else
                        put_message_with_wait
                              ( "Use a printable character please" );

                  }  /* End else */

            /* Tell the user what to do */
              put_message ( "Select a character from the fields displayed" );

            /* Just keep on looping til user wants to exit */
            } while ( NOT wait_for_user_event () );

      /* Fini */
      clear_message();
      return 0;
      }
```

```
int set_exemplar_data ( exemp_type * e )
/*
 *    This function gets the dot matrix and information for char number
 *    nchar of field number nfield, and inserts it in the exemplar set
 *    array.  If any portion of it fails, return is non-zero and the
 *    exemplar set is unmodified
 */
      {

      /* Go out and get the box dimensions for the dot matrix */
      get_selected_fieldchar_box ( &(e->sbox) );

      /* Got it.  Put in exemplar set */
      e->dm_ptr = get_dm ( &(e->sbox) );
      if ( e->dm_ptr == NULL )
            {
            /* Couldn't get the dot matrix */
            put_message_with_wait ( "Error: Couldn't get exemplar dot matrix"
);
            return -1;
            }

      /* Init the other data */
      e->cwidth  = e->sbox.pt2.x - e->sbox.pt1.x + 1;
      e->cheight = e->sbox.pt2.y - e->sbox.pt1.y + 1;
      e->drawn = 0;
      e->norm = 0;

      /* Fini */
      return 0;
      }



int add_exemplar ( exemp_type *  ex )
/*
 *    This function adds the exemplar character pointed to by ex to the
 *    exemplar set array.
 */
      {
      int i=-1, j;

      /* See if we can */
      if ( nexemp >= nexemp_allocated )
            {
            /* No more room to add.  Tell the user */
            put_message_with_wait
                  ( "Inadequate space: Please exit editor and start again");
            free ( ex->dm_ptr );
            return -1;
            }

      /* Find the position to add the character */
      do  {
            if ( ++i >= nexemp ) break;
            } while ( (exempset[i].ch) <= (ex->ch) );
```

```
        /* i now defines where we want to place the character */
        if ( i == nexemp )
              copy_exemplar ( ex, &(exempset[i]) );
        else
              {
              /* We need to move everything down one slot in the array */
              for ( j=nexemp; j >= i; j-- )
                    copy_exemplar ( &(exempset[j]), &(exempset[j+1]) );
              copy_exemplar ( ex, &(exempset[i]) );
              }

        /* Now update the number of exemplars */
        nexemp ++ ;

        fix_exemplar_box ( -1 );
        draw_exemplar ( -1 );

        /* Fini */
        return 0;
        }


void copy_exemplar ( exemp_type * src, exemp_type * dest )
        {
        /* Copy the basic data */
        dest->ch          = src->ch;
        dest->cwidth      = src->cwidth;
        dest->cheight     = src->cheight;
        dest->norm        = src->norm;
        dest->drawn       = src->drawn;
        dest->dm_ptr      = src->dm_ptr;

        /* Copy the point data */
        dest->ptchar.x  = src->ptchar.x;
        dest->ptchar.y  = src->ptchar.y;

        /* Copy the entire exemp box location */
        dest->ebox.pt1.x  = src->ebox.pt1.x;
        dest->ebox.pt1.y  = src->ebox.pt1.y;
        dest->ebox.pt2.x  = src->ebox.pt2.x;
        dest->ebox.pt2.y  = src->ebox.pt2.y;

        /* Copy the source box location */
        dest->sbox.pt1.x  = src->sbox.pt1.x;
        dest->sbox.pt1.y  = src->sbox.pt1.y;
        dest->sbox.pt2.x  = src->sbox.pt2.x;
        dest->sbox.pt2.y  = src->sbox.pt2.y;


        /* Fini */
        }


int delete_exemplar ( void )

        {
```

```
      int i;

      /* Loop until user is done */
      do    {
            /* Is an exemplar selected */
            if ( isexemplar_selected () )
                  {

                  /* Confirm user really wants to delete */
                        if ( confirm ( "Confirm to delete the character: ") >
0 )
                        {
                        undraw_exemplar ( exselect );

                        /* Kill the dot matrix allocation */
                        free ( exempset[exselect].dm_ptr );

                        /* Now copy the remaining characters down one */
                        nexemp = nexemp - 1;
                        for ( i = exselect; i<nexemp; i++ )
                              copy_exemplar ( &exempset[i+1], &exempset[i] );

                        /* Finally, unselect the exemplar */
                        unselect_exemplar();

                        } /* End confirm test */

                  else
                        unselect_exemplar();

                  } /* End isexemplar_selected test */

            /* Now tell the user what to do */
            put_message ( "Select the exemplar character to delete" );

            } while ( NOT wait_for_user_event() );

      /* Fini */
      return 0;
      }


int change_exemp_id ( void )
      {

      /* Loop until user is done */
      do    {

            /* See if an exemplar has been selected */
            if ( isexemplar_selected () )
                  {
                  /* Tell user what it is and let him change it */
                  sprintf ( ctbuf, "Current character id is: %c; Input new id:
",
                                    exempset[exselect].ch );
                  put_message ( ctbuf );
```

```
                /* Wait for user to identify the dot matrix */
                if ( wait_for_buffered_kbin(1) < 0 )
                      /* He hit escape, so break out of inner loop */
                      unselect_exemplar ( );

                /* See if char is printable */
                else if ( isprint ( kbin_buf[0] ) )
                      {
                      /* Got a valid key, so do it */
                      exempset[exselect].ch = kbin_buf[0];
                      draw_exemplar ( exselect );
                      unselect_exemplar();
                      }
                else
                      /* User hit a bad key, so tell him so */
                      put_message_with_wait ( "Warn: It must be a printable
char" );

                }     /* End isexemplar_selected test */


            /* Tell the user what to do */
            put_message ( "Select the exemplar character to change" );

            /* And wait for him to do something */
            } while ( NOT wait_for_user_event() );

       /* Fini */
       clear_message();
      return 0;
      }



int isexemplar_selected ( void )
      {
      int i, itemp;

      /* if there are no exemplars, return */
      if ( nexemp <= 0 )
            {
            exselect = -1;
            return 0;
            }

      /* Verify exselect is valid */
      if ( exselect >= nexemp ) exselect = -1;

      /* if the left button is down, see if the mouse is in an ex box */
      if ( buttons.left )
            {
            for ( i=0; i<nexemp; i++ )
                  {
                  itemp = ismouse_in_box_array ( &exempset[i].ebox, 1 );
                  if ( itemp == 0 )
                        exselect = i;
                  }
            }
```

```
	/* Now highlight the selected exemplar */
	if ( exselect != old_exselect )
		{
		if ( old_exselect >= 0 )
			undraw_box ( &exempset[old_exselect].ebox );
		draw_box ( &exempset[exselect].ebox );
		old_exselect = exselect;
		}

	/* If an exemplar is selected, return true */
	if ( exselect >= 0 ) return 1;
	else return 0;

	/* Thats it */
	}



void unselect_exemplar ( void )
	{
	/* undraw currently highlighted exemplar */
	if ( exselect >= 0 )
		undraw_box ( &exempset[old_exselect].ebox );

	/* and clear selector */
	old_exselect = exselect = -1;
	}


void undraw_exemplar ( int ex )
	{
	int e = ex, maxe = nexemp;

	/* set up the loop counter/limiter */
	if ( ex < 0 ) e = 0;
	else maxe = ex + 1;

	/* Now loop, undrawing the exemplar character */
	for ( ; e<maxe; e++ )
		{
		/* Only undraw if drawn */
		if ( exempset[ex].drawn )
			{
			/* Exemplar is drawn.  Clear the exemplar box */
			clear_box ( &exempset[ex].ebox );
			exempset[ex].drawn = 0;
			}	/* End drawn test */

		}	/* End for loop */

	/* Fini */
	}


void draw_exemplar ( int ex1 )
	{
	int e = ex1, maxe = nexemp;
```

```
        /* set up the loop counter/limiter */
        if ( ex1 < 0 ) e = 0;
        else maxe = ex1 + 1;

        /* Now loop, drawing the exemplar character(s) */
        for ( ; e<maxe; e++ )
              {
              /* Verify it fits in the exemplar window */
              if ( exempset[e].ebox.pt2.y < exemp_box.pt2.y )
                    {
                    /* OK to draw.  Put the dot matrix in sbox */
                    put_dm ( exempset[e].dm_ptr, &exempset[e].sbox );

                    draw_box_with_margin ( &exempset[e].sbox, 1 );

                    /* Display the char */
                    ctbuf[0] = exempset[e].ch;
                    ctbuf[1] = '\0';
                    out_text ( exempset[e].ptchar, ctbuf, 1, REVERSE_TEXT );

                    /* Now indicate it's drawn */
                    exempset[e].drawn = 1;

                    }     /* End ebox<exemp_box test */

              else
                    /* Can't draw exemplar, so indicate not drawn */
                    exempset[e].drawn = 0;

              }     /* End for loop */

        /* Fini */
        }


static void fix_exemplar_box ( int ex )
        {
        int e = ex, maxe = nexemp;
        int i, x, dx, centerx, y, dy, maxy;

        /* set up the loop counter/limiter */
        if ( ex < 0 ) e = 0;
        else maxe = ex + 1;

        for ( ; e<maxe; e++ )
              {

              /* Determine the starting x and y position */
              if ( e==0 )
                    {
                    x = exemp_box.pt1.x + MARGIN/2;
                    y = exemp_box.pt1.y + MARGIN/2;
                    }
              else
                    {
                    x = exempset[e-1].ebox.pt2.x + MARGIN/2;
                    y = exempset[e-1].ebox.pt1.y;
```

```
        }

    /* Determine the x size of ebox */
    dx = font.charwide;
    if ( dx < exempset[e].cwidth ) dx = exempset[e].cwidth;
    dx = dx + 8;

    /* See if the ebox will fit on current line */
    if ( ( x + dx + MARGIN/2 ) > exemp_box.pt2.x )
        {
        /* It won't fit on current line.  Find the largest y value
*/
        maxy = exemp_box.pt1.y;
        for ( i=0; i<e; i++ )
            {
            if ( maxy < exempset[i].ebox.pt2.y )
                maxy = exempset[i].ebox.pt2.y;
            }
        y = maxy + MARGIN/2;
        x = exemp_box.pt1.x + MARGIN/2;
        }
    dy = 4 + font.charhigh + 4 + exempset[e].cheight + 4;

    /* Now see if we need to modify the boxes */
    if (  ( exempset[e].ebox.pt1.x != x )
        || ( exempset[e].ebox.pt1.y != y )
        ||   ( exempset[e].ebox.pt2.x != x+dx-1 )
        ||   ( exempset[e].ebox.pt2.y != y+dy-1 ) )
        {
        /* Box must be changed.  Undraw it as required */
        if ( exempset[e].drawn )
            {
            undraw_exemplar ( e );
            exempset[e].drawn = 0;
            }

        /* Compute center x of the ebox */
        centerx = ( 2*x + dx ) / 2;

        /* Fix ebox */
        exempset[e].ebox.pt1.x = x;
        exempset[e].ebox.pt2.x = x+dx-1;
        exempset[e].ebox.pt1.y = y;
        exempset[e].ebox.pt2.y = y+dy-1;

        /* Fix point where character is placed */
        exempset[e].ptchar.x = centerx - font.charwide/2;
        exempset[e].ptchar.y = y + 4;

        /* Now define the exemplar dot matrix transfer box */
        exempset[e].sbox.pt1.x = centerx - exempset[e].cwidth/2;
        exempset[e].sbox.pt2.x = exempset[e].sbox.pt1.x
                                        + exempset[e].cwidth -
1;
        exempset[e].sbox.pt1.y = exempset[e].ptchar.y
                                        + font.charhigh + 3;
        exempset[e].sbox.pt2.y = exempset[e].sbox.pt1.y
```

```
                                                        + exempset[e].cheight -
1;

                }     /* End ebox tests */

            }     /* End for e loop */

      /* Fini */
      }



/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/

static BYTE threshold;
static int  cur_exemp;

/* apply threshold to exemplar editor buffer by computing the average
      pixel value in the dm and then setting each pixel to black or white
      if it was below or above the average
*/

static void thresh_exemplar_editor ( int e )
      {
      int i;
      unsigned long ave;
      BYTE *dm;

      dm = editbuffer;
      ave = 0;

      for ( i=0; i<exempset[e].cwidth*exempset[e].cheight; i++)
            ave += dm[i];

      ave = ave / ( exempset[e].cwidth * exempset[e].cheight );

      for ( i=0; i<exempset[e].cwidth*exempset[e].cheight; i++)
            if ( dm[i] > ave )
                  dm[i] = 255;
            else
                  dm[i] = 0;

      threshold = ave;
      cur_exemp = e;
      }


int increase_threshold ( void )
      {
      BYTE *dm;
      int i;

      threshold ++;
      dm = editbuffer;
```

```
      for ( i=0; i<exempset[cur_exemp].cwidth*exempset[cur_exemp].cheight;
i++)
            if ( exempset[cur_exemp].dm_ptr[i] > threshold )
                  dm[i] = 255;
            else
                  dm[i] = 0;

      display_exemplar ( -1 );
      return 0;
      }


int decrease_threshold ( void )
      {
      BYTE *dm;
   int i;

      threshold --;
      dm = editbuffer;

      for ( i=0; i<exempset[cur_exemp].cwidth*exempset[cur_exemp].cheight;
i++)
            if ( exempset[cur_exemp].dm_ptr[i] > threshold )
                  dm[i] = 255;
            else
                  dm[i] = 0;

      display_exemplar ( -1 );
      return 0;
      }


/*  return 1 if mouse is in editbox, 0 otherwise.  Also computes
       position of mouse in box coordinates.
*/

static int get_mouse_exemplar (int *x, int *y)
      {
      if  (    ( mouse_xy.x >= editbox.pt1.x )
            &&  ( mouse_xy.x <= editbox.pt2.x )
            &&  ( mouse_xy.y >= editbox.pt1.y )
            &&  ( mouse_xy.y <= editbox.pt2.y ) )
           {
                  *x = mouse_xy.x - editbox.pt1.x - 1;
                  *y = mouse_xy.y - editbox.pt1.y - 1;
                  return 1;
           }
      else
      if  (    ( mouse_xy.x >= orgbox.pt1.x )
            &&  ( mouse_xy.x <= orgbox.pt2.x )
            &&  ( mouse_xy.y >= orgbox.pt1.y )
            &&  ( mouse_xy.y <= orgbox.pt2.y ) )
           {
                  *x = mouse_xy.x - orgbox.pt1.x - 1;
                  *y = mouse_xy.y - orgbox.pt1.y - 1;
                  return 1;
           }
```

```
        else
                return 0;

        }


/*  erase the ememplar and editbuffer from screen   */
static void erase_exemplar ( void )
        {
        clear_box ( &orgbox );
        delay ( 5 );
        clear_box ( &editbox );
        }

/*  compute the boxes used for drawing the exemplar and the edit buffer */

static void scale_exemplar (void)
        {
        int x, y;

        x = exemp_mag * exempset[exselect].cwidth;
        y = exemp_mag * exempset[exselect].cheight;

        if ( x >= 1000 )
        {
                exemp_mag --;
                return;
        }

        /* the edit box */
        editbox.pt1.x = display_size.x / 2 + display_size.x / 4 - x / 2 ;
        editbox.pt2.x = editbox.pt1.x + x ;
        editbox.pt1.y = display_size.y / 2 - y / 2 ;
        editbox.pt2.y = editbox.pt1.y + y ;

        /* the original exemplar box */
        orgbox.pt1.x = display_size.x / 4 - x / 2 ;
        orgbox.pt2.x = orgbox.pt1.x + x ;
        orgbox.pt1.y = display_size.y / 2 - y / 2 ;
        orgbox.pt2.y = orgbox.pt1.y + y ;
        }


/* Draw the original exemplar and the editbuffer side by side and magnify
        according to the value of exemp_mag.
*/
static void display_exemplar (int therow)
        {
        int i, j, mj, mi, x, y, startrow;
        BYTE row[1000];
        box_type ebox;                  /* the current row of the editbox */
        box_type obox;                          /* the current row of the exemplar
box */

        x = exempset[exselect].cwidth;
        y = exempset[exselect].cheight;
```

```
	/* set up the widths of the boxes */
	ebox.pt1.x = editbox.pt1.x;
	ebox.pt2.x = editbox.pt2.x;

	obox.pt1.x = orgbox.pt1.x;
	obox.pt2.x = orgbox.pt2.x;

	memset ( row, 0, 1000 );

	/* if therow is > 0 then I only need to draw that row. */
	if ( therow >= 0 )
	{
		/* set up the y coords of the drawbox for the desired row */
		startrow = therow;
		y = therow + 1;

		ebox.pt1.y = ebox.pt2.y = editbox.pt1.y + ( therow * exemp_mag );
		obox.pt1.y = obox.pt2.y = orgbox.pt1.y + ( therow * exemp_mag );
	}
	else
	{
		/* therow was < 0 so I need to draw the whole image */
		startrow = 0;
		ebox.pt1.y = ebox.pt2.y = editbox.pt1.y;
		obox.pt1.y = obox.pt2.y = orgbox.pt1.y;
	}

	for ( j = startrow; j < y; j ++ )  /* for each row */
		{
			/* draw original */
			for ( i = 0; i < x; i ++ ) /* for each pixel in the original
*/
				for ( mi = 0; mi < exemp_mag; mi ++ ) /* magnify
column */
					/* fill row with the pixels */
					row [ i * exemp_mag + mi ] =
exempset[exselect].dm_ptr [ j * x + i ];

			for ( mj = 0; mj < exemp_mag; mj ++ ) /* magnify rows */
			{
				/* draw row */
				put_dm ( row, &obox );
				obox.pt1.y ++;
				obox.pt2.y ++;
			}

			/* draw edit buffer */
			for ( i = 0; i < x; i ++ )
				for ( mi = 0; mi < exemp_mag; mi ++ )
					row [ i * exemp_mag + mi ] = editbuffer [ j * x
+ i ];

			for ( mj = 0; mj < exemp_mag; mj ++ )
			{
				put_dm ( row, &ebox );
				ebox.pt1.y ++;
				ebox.pt2.y ++;
			}
```

```
            }
        draw_box ( &editbox );
        draw_box ( &orgbox );
        }

/* change the pixel fill color */
int toggle_exemplar_color ( void )
        {
        if ( fillcolor )
                fillcolor = 0;
        else
                fillcolor = 255;
        return 0;
        }


int magnify_exemplar ( void )
        {
        exemp_mag ++;
        erase_exemplar ();
        scale_exemplar ();
        display_exemplar ( -1 );
        return 0;
        }

int demagnify_exemplar ( void )
        {
        exemp_mag --;
        if ( exemp_mag == 0 )
                exemp_mag = 1;
        erase_exemplar ();
        scale_exemplar ();
        display_exemplar ( -1 );
        return 0;
        }


static void set_exemplar_pixel ( int x, int y, BYTE val ) /* x & y in editbox
coords */
{
        editbuffer [ ( y / exemp_mag ) * exempset[exselect].cwidth + ( x /
exemp_mag)] = val;
        display_exemplar ( (int ) ( y / exemp_mag ) );
}


static void edit_exemplar ( void )
        {
        int x, y;


        /* clear screen */
        clear_display ();
        clear_message ();


        /* make copy of the selected exemplar's dm */
```

```
      editbuffer = ( BYTE * ) malloc ( exempset[exselect].cwidth *
exempset[exselect].cheight );
      memcpy ( editbuffer, exempset[exselect].dm_ptr,
exempset[exselect].cwidth * exempset[exselect].cheight );

      /* apply threshold to copy */
      thresh_exemplar_editor (exselect);

      /* initialize magnification to arbitrary value */
      exemp_mag = 14;

      /* draw edit screen */
      put_screen_title ( "EXEMPLAR CHARACTER EDITOR" );
      scale_exemplar ();
      display_exemplar ( -1 );

      /* Loop until user is done */
      do    {
            enable_command ( &magex );
            enable_command ( &demagex );
            enable_command ( &incthresh );
            enable_command ( &decthresh );
            enable_command ( &fillcol );

            if ( buttons.left ) /* is button pushed */
                  if ( get_mouse_exemplar ( &x, &y ) )   /* is in edit box */
                        set_exemplar_pixel ( x, y, fillcolor); /* fill pixel
*/

            } while ( NOT wait_for_user_event() );

      if ( confirm ( "Keep changes to exemplar character ? ") > 0 )
            {
            /* replace exmplars dm with the edit buffer */
            free (exempset[exselect].dm_ptr );
            exempset[exselect].dm_ptr = editbuffer;
            }
      else
            /* ditch the edit buffer */
            free ( editbuffer );

      /* restore the exemplar select screen */
      clear_display ();
      draw_exemplar_id_screen ();

      }


int edit_exset ( void )
      {

      /* Loop until user is done */
      do    {
            /* Is an exemplar selected */
            if ( isexemplar_selected () )
                  {
                  edit_exemplar ( );
```

```
                } /* End isexemplar_selected test */

            /* Now tell the user what to do */
            put_message ( "Select the exemplar character to edit" );

            } while ( NOT wait_for_user_event() );

   unselect_exemplar ();
      /* Fini */
      return 0;
      }
```

EXEMPSZ.C

```
#include <ctype.h>
#include <conio.h>
#include <stdlib.h>
#include <malloc.h>
#include <dos.h>
#include <mem.h>

#include "net.h"
#include "mouse.h"
#include "vcrsmfg.h"
#include "ntiga.h"



static int xsize = 20;
static int ysize = 25;

/* Declare local functions */
static int        init_exemplar_sizer  ( void );
static void       exit_exemplar_sizer  ( void );
static void get_best_size ( int *, int * );


static box_type exsz_box;

static void draw_exemplar ( void );
static void scale_exemplars ( void );


int exemplar_sizer  ( void )
      {
      int itemp, modified;
      char str[155];

      if ( init_exemplar_sizer () ) return -1;
      get_best_size ( &xsize, &ysize );
      put_screen_title ( "EXEMPLAR SET SIZING" );

      put_box_title ( "CURRENT EXEMPLARS", &exsz_box );
      draw_exemplar ();

      /* Command Processing Loop */
      do
            {
```

```
            modified = 1;

            itemp = check_edit_arrows();
            if      ( itemp == UP_ARROW )       ysize++;
            else if ( itemp == DOWN_ARROW )     ysize--;
            else if ( itemp == RIGHT_ARROW )    xsize++;
            else if ( itemp == LEFT_ARROW )     xsize--;
            else modified = 0;

            sprintf (str, "L/R inc/dec width, U/D inc/dec height. w = %d h =
%d\0",
                  xsize, ysize);
            put_message ( str );

            if ( modified )
                  {
                  if ( xsize < 1 ) xsize = 1;
                  if ( ysize < 1 ) ysize = 1;
                  clear_box ( &exsz_box );
                  draw_exemplar ();
                  }


            } while ( NOT wait_for_user_event() );

      /* Save the data */
      exit_exemplar_sizer  ();

      /* Fini */
      return 0;
      }


int init_exemplar_sizer  ( void )
      {
      box_type arrows_box;

      /* Now set up the exemplar set box. Do x'es */
      exsz_box.pt1.x = MARGIN;

      exsz_box.pt2.x = display_size.x - MARGIN;

      /* Now do y's */
      exsz_box.pt1.y = 500;
      exsz_box.pt2.y = display_size.y - MARGIN - 200;

      arrows_box.pt1.x = 100;
      arrows_box.pt2.x = 400;
      arrows_box.pt1.y = 100;
      arrows_box.pt2.y = 400;
      init_arrows_edit_box ( &arrows_box );

      /* Go get the field data, and set up field boxes */
      if ( get_field_set() )
            /* Field data is bad.  Major stake in the heart */
            return -1;
```

```
      /* Now get existing exemplar data, if it exists */
      if ( get_exemplar_set ( filexemp ) ) return -1;

      /* Fini */
      return 0;
      }



void exit_exemplar_sizer  ( void )
      {

      /* See if we need to save the raw and normalized exemplar data */
      if ( ( ! fexist ( filtrainset ) ) || ( fexist ( filtrainset ) &&
                              ( confirm ( "Do you want to save training set
?" ) > 0 ) ) )
            {
            scale_exemplars ();
            save_exemp_as_train ( filtrainset );
            }


      /* Now destroy dynamically allocated stuff */
      kill_exemplar_set ( );
      kill_field_set ();
      kill_arrow_boxes ();
      /* Clear the display for the main application */
      clear_display ();

      /* Fini */
      }



static void scale_exemplars ( void )
      {
      int i, j, e;
      int ox, oy;
      BYTE *olddm, *newdm;
      float yfact, xfact;
      char str[85];

      for ( e=0; e < nexemp; e++ )
            {
            olddm = exempset[e].dm_ptr;
            newdm = ( BYTE * ) malloc ( xsize * ysize );
            if ( !newdm )
                  {
                  sprintf (str,"%s %d %d %d %d\0","Scale Malloc error", e,
nexemp, xsize, ysize);
                  put_message_with_wait (str);
                  return;
                  }

            xfact = ((float)xsize / (float)exempset[e].cwidth);
            yfact = ((float)ysize / (float)exempset[e].cheight);
```

```
            for ( j=0; j < ysize; j++ )
                  {
                  oy = (int) ((float)j) / yfact;
                  for ( i=0; i < xsize; i++ )
                        {
                        ox = (int) ((float)i) / xfact;
                        newdm [ j * xsize + i ] = olddm
[oy*exempset[e].cwidth+ox];
                        }
                  }
            exempset[e].cwidth = xsize;
            exempset[e].cheight = ysize;
            free ( olddm );
            exempset[e].dm_ptr = newdm;
            }
      }


static void draw_exemplar ( void )
      {
      int i, j, e;
      int ox, oy;
      BYTE *olddm, newdm[150];
      float yfact, xfact;
      box_type sbox;
      int dx, dy;

      dx = exsz_box.pt1.x + MARGIN/2;
      dy = exsz_box.pt1.y + MARGIN/2;

      memset ( newdm, 0, 150 );

      for ( e=0; e < nexemp; e++ )
            {
            olddm = exempset[e].dm_ptr;

            xfact = ((float)xsize / (float)exempset[e].cwidth);
            yfact = ((float)ysize / (float)exempset[e].cheight);

            sbox.pt1.x = dx;
            sbox.pt2.x = dx + xsize - 1;

            for ( j=0; j < ysize; j++ )
                  {
                  oy = (int) ((float)j) / yfact;
                  for ( i=0; i < xsize; i++ )
                        {
                        ox = (int) ((float)i) / xfact;
                        newdm [ i ] = olddm [oy*exempset[e].cwidth+ox];
                        }

                  sbox.pt1.y = dy + j;
                  sbox.pt2.y = dy + j;

                   put_dm (newdm, &sbox);

                   }
            dx += xsize + MARGIN/2;
```

```
            }
      }


void get_best_size ( int * x, int * y )
      {
      int e;
      unsigned long ax=0, ay=0;

      for ( e=0; e < nexemp; e++ )
            {
            ax += exempset[e].cwidth;
            ay += exempset[e].cheight;
            }

      *x = (int) ax/nexemp;
      *y = (int) ay/nexemp;
      }


/*************************************************************************
*
*   FBOXES.C
*
*************************************************************************
*/

#include <malloc.h>

#include "mouse.h"
#include "net.h"
#include "ntiga.h"
#include "vcrsmfg.h"

/* Declare local states */
static int high_field, high_char;
static int fselect=-1, cselect=-1;


#define FXMARGIN 10
#define FYMARGIN 10



void fix_field_boxes ( int fn, int draw_flag )
/*  The field number to be fixed is provided by fn.  if fn<0, fix all */
      {
      int j, fnmax = fn+1, cfn = 0;
      short nchars, centerx, wdb, fw, x, y, mcw, mch, chars_per_line;
      point_type * p;
      box_type *s, *d;

      /* If we got no fields, get outa here */
      if ( nfields <= 0 ) return;
      /* otherwise, initialize the loop counter and limiter */
      else if ( fn < 0 ) fnmax = nfields;
      else cfn = fn;
```

```
        /* Determine center and width of the display region */
        centerx = ( field_box.pt1.x + field_box.pt2.x ) / 2;
        wdb = field_box.pt2.x - field_box.pt1.x + 1;

        /* Now loop through the fields as required */
        do  {
              /* Undraw the field */
              undraw_field ( cfn );

              /* Set up local variables for this field */
              nchars = fieldset[cfn].nchars;
              s = fieldset[cfn].sbox;
              d = fieldset[cfn].dbox;
              p = fieldset[cfn].points;
              mcw = mag * fieldset[cfn].cwidth;
              mch = mag * fieldset[cfn].cheight;

              /* Set up y's */
              if ( cfn == 0 ) y = field_box.pt1.y + FYMARGIN / 2;
              else y = fieldset[cfn-1].fdbox.pt2.y + FYMARGIN;

              /* Determine field width per line */
              fw = (nchars * ( mcw + FXMARGIN )) + (2 * FXMARGIN);
              chars_per_line = nchars;
              if ( fw > wdb )
                    {
                    chars_per_line = (( wdb - 2 * FXMARGIN ) / ( mcw + FXMARGIN
));
                    fw = (chars_per_line) * ( mcw + FXMARGIN ) + (2*FXMARGIN);
                    }

              /* Set up the upper left coordinates for fields' box */
              fieldset[cfn].fdbox.pt1.x = centerx - (fw / 2);
              fieldset[cfn].fdbox.pt1.y = y;
              x = fieldset[cfn].fdbox.pt1.x + 3 * FXMARGIN / 2;
              y = y + FYMARGIN;

              /* Now loop thru the characters */
              for ( j=0; j<nchars; j++ )
                    {

                    /* Initialize the character source box */
                    s[j].pt1.x = p[j].x + frame_box.pt1.x;
                    s[j].pt1.y = p[j].y + frame_box.pt1.y;
                    s[j].pt2.x = s[j].pt1.x + fieldset[cfn].cwidth - 1;
                    s[j].pt2.y = s[j].pt1.y + fieldset[cfn].cheight - 1;

                    /* Now initialize the destination box data */
                    d[j].pt1.x = x;
                    d[j].pt1.y = y;
                    d[j].pt2.x = d[j].pt1.x + mcw - 1;
                    d[j].pt2.y = d[j].pt1.y + mch - 1;

                    /* Increment x and see if we need a new line */
                    x = x + mcw - 1 + FXMARGIN;
                    if ( ( (j+1) % chars_per_line == 0 ) && ( j+1 != nchars ) )
                          {
                          /* needum new line */
```

```
                        /* Reset x and y for new line start coordinates */
                        x = fieldset[cfn].fdbox.pt1.x + 3*FXMARGIN/2;
                        y = y + mch - 1 + FYMARGIN;
                        /* and don't trounce other parts of the display */
                        if ( y > ( field_box.pt2.y - FYMARGIN/2 ) ) break;
                        }   /* End x test */

                  }   /* End nchars loop */

            /* Set lower y for the field */
            fieldset[cfn].fdbox.pt2.x = centerx + (fw/2);
            fieldset[cfn].fdbox.pt2.y = y + mch - 1 + FYMARGIN;

            /* And set up the box to be drawn in the frame */
            fieldset[cfn].fsbox.pt1.x = s[0].pt1.x;
            fieldset[cfn].fsbox.pt2.x = s[nchars-1].pt2.x;
            fieldset[cfn].fsbox.pt1.y = s[0].pt1.y;
            fieldset[cfn].fsbox.pt2.y = s[nchars-1].pt2.y;

            /* Now draw the field */
            if (draw_flag)
                  draw_field ( cfn );

            /* See if we need to do the rest of the fields */
            if ( ( fnmax < nfields ) && ( cfn != nfields-1 ) )
                  {
                  /* See if field boxes are apportioned correctly */
                  if ( ( fieldset[cfn].fdbox.pt2.y + FYMARGIN/2 + 1 )
                        != ( fieldset[cfn+1].fdbox.pt1.y ) )
                        /* If we get here, we need to do the rest of the
fields */
                        fnmax = nfields;
                  }

            } while ( ++cfn < fnmax );  /* End do */

      /* All done */
      }


void highlight_field ( int fn )
      {
      /* if any field is highlighted, unhighlight it */
      if ( high_field != -1 ) unhighlight_field();

      /* Now highlight the field selected */
      high_field = fn;
      draw_box_with_margin ( &fieldset[fn].fdbox, 2 );
      draw_box_with_margin ( &fieldset[fn].fdbox, 3 );
      }


void highlight_field_char ( int fn, int cn )
      {
      /* if any char is highlighted, unhighlight it */
      if ( high_char != -1 ) unhighlight_field_char();

      /* Verify field selected is highlighted */
```

```
      if ( fn != high_field ) highlight_field ( fn );

      /* Now highlight the char selected */
      high_char = cn;
      draw_box_with_margin ( &fieldset[fn].dbox[cn], 2 );
      draw_box_with_margin ( &fieldset[fn].dbox[cn], 3 );
/*    draw_box ( &fieldset[fn].sbox[cn] );*/
      }


void unhighlight_field ( void )
      {
      /* If there is a highlighted field box, undraw it */
       if ( high_field != -1 )
              {
              undraw_box_with_margin ( &fieldset[high_field].fdbox, 2 );
              undraw_box_with_margin ( &fieldset[high_field].fdbox, 3 );
              }

      /* Now indicate no field is highlighted */
      high_field = -1;
      }


void unhighlight_field_char ( void )
      {
      /* If there is a highlighted char box, undraw it */
       if ( high_char != -1 )
              {
              undraw_box_with_margin ( &fieldset[high_field].dbox[high_char],
2 );
              undraw_box_with_margin ( &fieldset[high_field].dbox[high_char],
3 );
/*            undraw_box ( &fieldset[high_field].sbox[high_char] ); */
              }

      /* Now indicate no char is highlighted */
      high_char = -1;
      }


int  ismouse_in_field_box ( void )
       {
       int cfn;   /* current field number */

       /*  Check mouse against each field box */
       for ( cfn=0; cfn<nfields; cfn++ )
              {
            if  (   ( mouse_xy.x >= fieldset[cfn].fdbox.pt1.x )
                  &&  ( mouse_xy.x <= fieldset[cfn].fdbox.pt2.x )
                  &&  ( mouse_xy.y >= fieldset[cfn].fdbox.pt1.y )
                  &&  ( mouse_xy.y <= fieldset[cfn].fdbox.pt2.y ) )

                  /* This path if we got the field */
                  return cfn;

              }   /* End cfn loop */
```

```
        /* mouse is not in any of the field boxes */
        return -1;
        }

int  ismouse_in_fieldchar_box ( int fn )
        {
        int i;

        /* Verify fn is reasonable */
        if ( ( fn < 0 ) || ( fn >= nfields ) ) return -1;

       i = ismouse_in_box_array ( fieldset[fn].dbox, fieldset[fn].nchars );
       return i;
       }


void draw_field ( int fn )
/*
*      This function draws the field selected by fn.  If fn < 0, draw
*      all fields.  There is no return.
*/
       {
       int cfn = 0, fnmax = fn+1, j;

       /* Initialize for entry into the loop, either one field or all */
       if ( fn < 0 )      fnmax = nfields;
       else               cfn = fn;

       /* Now start looping through each field */
       for ( ; cfn<fnmax; cfn++ )   /* Note that cfn was initialized above */
             {
             /* If the field is drawn, undraw it */
               if ( fieldset[cfn].drawn )
                         undraw_field ( cfn );

             /* Loop thru each character in the field */
             for ( j=0; j<fieldset[cfn].nchars; j++ )
                   {
                   /* Blit the source from the video frame to the field box */
                         zoom_box ( &fieldset[cfn].sbox[j],
&fieldset[cfn].dbox[j] );

                   /* Now draw the border around the char box */
                         draw_box_with_margin ( &fieldset[cfn].dbox[j], 1 );
                   }   /* End nchars loop */

             /* Now draw the border around the field and in the frame */
             draw_box ( &fieldset[cfn].fdbox );

             /* Indicate the field has been drawn */
             fieldset[cfn].drawn = 1;

             }   /* End fields loop */

       /* Fini */
       }
```

```
void undraw_field ( int fn )
/* Blottos the field specified.  If fn < 0, blotto all fields */
      {
       int cfn = 0, fnmax = fn+1;

      if ( fn<0 ) fnmax = nfields;
      else        cfn = fn;

      for ( ; cfn<fnmax; cfn++ )
            {
            /* Verify the field is drawn */
            if ( fieldset[cfn].drawn )
                  {
                  /* Blotto the fields' box */
                  clear_box ( &fieldset[cfn].fdbox );

                  /* And indicate field is not drawn */
                  fieldset[cfn].drawn = 0;

                  }     /* End drawn test */

            }     /* End for cfn loop */

      /* That's it */
      }


void draw_frame_field ( int fn , unsigned long col, unsigned int ppop )
/*
*     This function draws the field in the frame box selected by fn.  If fn <
0, draw
*     all fields.  There is no return.
*/
      {
      int cfn = 0, fnmax = fn+1;
      unsigned int oldppop;
      unsigned long oldcol;

      /* Initialize for entry into the loop, either one field or all */
      if ( fn < 0 )     fnmax = nfields;
      else                    cfn = fn;

      oldppop = get_ppop ( );
      set_ppop ( ppop );
      /* Now draw the border around the field and in the frame */
  oldcol = fcolor;
      fcolor = col;
      for ( ; cfn<fnmax; cfn++ )    /* Note that cfn was initialized above */
            draw_box ( &fieldset[cfn].fsbox );
      set_ppop ( oldppop );
      fcolor = oldcol;
      /* Fini */
      }


void draw_frame_char ( int fn, int cn , unsigned long col, unsigned int ppop )
      {
      int cfn = 0, fnmax = fn+1, j;
```

```
      int cnmax, tcn;
      unsigned int oldppop;
      unsigned long oldcol;

      /* Initialize for entry into the loop, either one field or all */
      if ( fn < 0 )     fnmax = nfields;
      else              cfn = fn;

      /* Now start looping through each field */
      for ( ; cfn<fnmax; cfn++ )   /* Note that cfn was initialized above */
            {

            if ( cn < 0 )
                  {
                  tcn = 0;
                  cnmax = fieldset[cfn].nchars;
                  }
            else
                  {
                  tcn = cn;
                  cnmax = cn + 1;
                  }

            oldppop = get_ppop ( );
            set_ppop ( ppop );

            oldcol = fcolor;
            fcolor = col;
            /* Now draw the border around the char box */
            for ( j=tcn; j<cnmax; j++ )
                        draw_box ( &fieldset[cfn].sbox[j] );
            set_ppop ( oldppop );

            fcolor = oldcol;
            }   /* End fields loop */

      /* Fini */
      }


void undraw_frame_char ( int fn, int cn , unsigned long col, unsigned int ppop
)
      {
      draw_frame_char ( fn, cn, col, ppop );  /* since XOR mode is used undraw
= draw */
      }


void undraw_frame_field ( int fn , unsigned long col, unsigned int ppop )
      {
      draw_frame_field ( fn, col, ppop );  /* since XOR mode is used undraw =
draw */
      }


void change_field_char ( int fn, int cn )
/*
*  This function modifies the box data associated with a character
```

```
*  number cn of field number fn.
*/
      {
       box_type * s;


      /* Initialize s to point at the source box for this field */
       s = &fieldset[fn].sbox[cn];
       undraw_frame_char ( fn, cn, 0xff, 10 );

      /* Now, modify the source box */
       s->pt1.x = fieldset[fn].points[cn].x + frame_box.pt1.x;
       s->pt1.y = fieldset[fn].points[cn].y + frame_box.pt1.y;
       s->pt2.x = s->pt1.x + fieldset[fn].cwidth - 1;
       s->pt2.y = s->pt1.y + fieldset[fn].cheight - 1;

      /* Now zoom the source box dot matrix into the destination box */
       zoom_box ( s, &fieldset[fn].dbox[cn] );

       draw_frame_char ( fn , cn, 0xff, 10 );

      /* Fini */
      }


int isfieldchar_selected ( void )
      {
      int ftemp, ctemp;

       /* if the left button is down, see if the mouse is in a field box */
      if ( buttons.left )
            {
            ftemp = ismouse_in_field_box ();
       ctemp = ismouse_in_fieldchar_box ( ftemp );
       if ( ( ftemp >= 0 ) && ( ctemp >= 0 ) )
                 {
           /* A new field/char is selected */
           fselect = ftemp;
           cselect = ctemp;
           highlight_field_char ( fselect, cselect );
           return 1;
           }   /* End ftemp/ctemp test */

       }   /* End buttons test */

   /* If we get here, no character is selected */
       fselect = cselect = -1;
   unhighlight_field ( );
   return 0;
      }



void get_selected_fieldchar_box ( box_type * box )
/*
*     This function provides the source box data associated with the
```

```
*     selected field's character.
*/
      {
      /* Just copy the data from sbox to box */
      box->pt1.x = fieldset[fselect].sbox[cselect].pt1.x;
      box->pt1.y = fieldset[fselect].sbox[cselect].pt1.y;
      box->pt2.x = fieldset[fselect].sbox[cselect].pt2.x;
      box->pt2.y = fieldset[fselect].sbox[cselect].pt2.y;

      /* Fini */
      }


/*****************************************************************************
 *
 *    FIELD.C
 *
 *    This file contains the application for defining the fields of data
 *    to be passed to the neural network for recognition processing.  The
 *    principal function is field_editor, which provides the command
 *    shell for all field operations.  This portion of the application
 *    should only be invoked once the frame has been edited, which customizes
 *    for the specific video source.  The field editor saves all data
 *    automatically on exit, in a file named xxxxxxx.FLD, where xxxxxxx
 *    is either TRIAL or something the user defined during application
 *    load.
 *
 *****************************************************************************
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "mouse.h"
#include "net.h"
#include "ntiga.h"
#include "vcrsmfg.h"

/* Declare globals */
int nfields;                    /* # fields currently defined */
int nfields_allocated;      /* # fields currently allocated */
field_type * fieldset;      /* pointer to fields array */
box_type field_box;                   /* field data is presented here */

/* Declare local functions */
static int   init_field_editor ( void );
static void  exit_field_editor ( void );
static int   field_grab ( void );
static int   new_field ( void );
static int   change_magnification ( void );
static int   reorder_fields ( void );
static int   select_field ( void );
static int   change_char_size ( void );
static int   change_number_chars ( void );
static int   move_char ( void );
static int   delete_field ( void );
static short ftos ( float );
```

```
static void  copy_field_data ( field_type * , field_type * );
static int   fix_field_allocation ( field_type * );


/* Declare local file data */
static box_type arrows_box;                /* where editing arrows are
presented */
static int fselect;                              /* current field selected */
static int cselect;                              /* current char selected */

/* Here are the commands which may be issued within this module */

static command_type grab      =
      {     "GrabFrame",
            field_grab,
            "Grab a new video frame" };

static command_type new     =
      {     "NewField",
            new_field,
        "Define a new field for recognition processing" };

static command_type reorder   =
      {     "Reorder",
            reorder_fields,
            "Change the order of the fields" };

static command_type magnify   =
      {     "ChangeMag",
            change_magnification,
            "Change magnification of field data" };

static command_type selfld    =
      {     "SelField",
            select_field,
            "Select a field for editing" };

static command_type csize   =
      {     "CharSize",
            change_char_size,
            "Change char width or height for selected field" };

static command_type nchars  =
      {     "# Chars",
            change_number_chars,
            "Change number of char's in selected field" };

/*static command_type selchar =
      {     "MoveChar",
            move_char,
            "Move a character in selected field" };
*/
static command_type delete  =
      {     "DelField",
            delete_field,
            "Delete selected field" };
```

```
/*****************************************************************************
 *  Here is the principal routine called from netsetup.
 *****************************************************************************
 */

int field_editor ( void )
      {
      int temp;

      /* Initialize the field editor data */
      if ( init_field_editor() ) return -1;

      /* Loop until user causes exit */
      do {

            /* Init fselect and cselect */
        fselect = cselect = -1;
        unhighlight_field ();
        unhighlight_field_char ();

            /* if left button is down, user may be selecting a field box */
            if ( buttons.left )
                  {
                  /* See if mouse is in a field box */
                  temp = ismouse_in_field_box ( );
                  if ( temp >= 0 )
                        {
                        /* User has selected a field box, so do it */
                        fselect = temp;
                        if ( select_field() < 0 ) break;
                        }

                  /* if in the frame box, start a new field */
                  else if ( ismouse_in_box_array ( &frame_box, 1 ) >= 0 )

                        /* User is in there so do it */
                        if ( new_field() < 0 ) break;

                  }     /* End buttons.left test */

            /* Now, Set up and display commands */
            enable_command ( &grab );
            enable_command ( &new );
            enable_command ( &magnify );
            /* User can only select a field for editing if one exists */
            if ( nfields > 0 )
                  enable_command ( &selfld );

            /* Now, wait for the user to do something */
            } while ( NOT wait_for_user_event() );

      /* Call the exit routine, which will save the data as required */
      exit_field_editor ();

      /* Fini */
      return 0;
      }
```

```
int init_field_editor ( void )
/*
 *  This function initializes the field editor.  It sets up the boxes
 *  used to display information, gets existing field data (if it exists),
 *  sets up display components, and puts up the display
 */
     {

     /* Set up the box where field data will go */
     field_box.pt1.x = frame_box.pt2.x + MARGIN;
     field_box.pt1.y = frame_box.pt1.y;
     field_box.pt2.x = display_box.pt2.x;
     field_box.pt2.y = display_box.pt2.y;

     /* Now set up the box where the editing arrows will go */
     arrows_box.pt1.x = frame_box.pt1.x;
     arrows_box.pt2.x = frame_box.pt2.x;
     arrows_box.pt1.y = frame_box.pt2.y + MARGIN + font.charhigh;
   arrows_box.pt2.y = display_box.pt2.y;

     /* Get existing field data.  If there's none, allocate memory */
     if ( get_field_set () ) exit ( -1 );

     /* Init complete.  Draw the field editor screen */
     clear_display ();
     put_screen_title ( "FIELD EDITOR" );
     put_box_title ( "CURRENT FRAME", &frame_box );
     put_box_title ( "CURRENT FIELDS", &field_box );
     init_arrows_edit_box ( &arrows_box );

     /* Now grab a frame and display existing field boxes */
     put_message ( "Grabbing a frame" );
     get_frame (  );
     clear_message ( );
     fix_field_boxes ( -1, 1 );
     draw_box ( &frame_box );

     return 0;
     }


void exit_field_editor ( void )
/*
 *  This function provides a graceful exit from the field editor.  It
 *  will save the field data (after asking user), then kills the various
 *  dynamically allocated components required for field data display
 */
   {

     /* Automatically save if the file does not exist */
      if ( confirm ( "Do you want to save the field data ?" ) > 0 )
       save_field_set();

     /* Now kill field data, and display boxes */
     kill_field_set ();
     kill_arrow_boxes();
```

```
      /* clear the display and return */
      clear_display();

    /* Fini */
      }


int field_grab ( void )
      {

      /* get frame returns non-zero if a problem occurred */
      get_frame (  );

      /* Clear and redraw all field boxes */
      clear_box ( &field_box );
      draw_field ( -1 );

      clear_message ();

      draw_box ( &frame_box );

      /* Fini */
      return 0;
      }


int new_field ( void )
/* This function is the only way to define a field */
      {
      point_type * p;
      box_type * b;
      int itemp, i, nchars;
      float fx, fbias;
      unsigned int oldppop;

      /* Tell the user what to do */
      put_message ( "Drag left button to define the new field" );

      /* Now wait for user to do something */
      do  {
            if ( wait_for_user_event () ) return 0; /* abort */
            itemp = ismouse_in_box_array ( &frame_box, 1 );
              } while ( NOT ( buttons.left && ( itemp == 0 ) ) );

      /*
       *    Left is down in the frame box.  Tell the mouse driver
       *    to establish the fix point and drag box around.  When user
       *    releases the left button, we get a pointer to the box coordinates
       */
      put_message ( "Drag mouse to define the field" );
      b = mouse_fixpt_box ( );

      oldppop = get_ppop ( );
      set_ppop ( 10 );
      draw_box ( b );
```

```
      set_ppop ( oldppop );

      /* Get the number of chars in the field */
      nchars = get_number_from_user("Input the number of chars in the field:
");
      if ( nchars < 0 )
            {
            set_ppop ( 10 );
            draw_box ( b );
            set_ppop ( oldppop );
            clear_message ( );
            return 0; /* User wants to abort */
            }

      /*  We're set.  Take a cut at char width, and height */
      set_ppop ( 10 );
      draw_box ( b );
      set_ppop ( oldppop );

      fieldset[nfields].nchars = nchars;
      fieldset[nfields].cwidth = ( b->pt2.x - b->pt1.x + 1 ) / nchars;
      fieldset[nfields].cheight = b->pt2.y - b->pt1.y + 1;

      /* Now allocate for the point array */
      if ( allocate_field_arrays ( &fieldset[nfields] ) )
            /* Couldn't allocate */
            {
            put_message_with_wait ( "Error: Could not allocate field array" );
            return -1;
            }

      /* Initialize the point array */
      p = fieldset[nfields].points;
      fbias = (float)( b->pt2.x - b->pt1.x + 1 ) / (float) nchars;
      fx = (float) ( b->pt1.x - frame_box.pt1.x );
      for ( i=0; i<nchars; i++ )
            {
            p[i].x = ftos ( fx );
            p[i].y = b->pt1.y - frame_box.pt1.y;
            fx += fbias;
            }

      /* Increment the number of fields and set up all field boxes */
      undraw_field ( -1 );
      nfields ++;
      fix_field_boxes ( -1, 1 );

      /* See if user can still define fields */
      if ( ( nfields + 1 ) > nfields_allocated )
            {
            /* Don't have anymore allocated */
            put_message_with_wait ( "You've reached the max number of fields"
);
            put_message_with_wait ( "Exiting the field editor.  You may
reenter to define more fields" );
            return -1;
            }
```

```
      /* Fini */
      return 0;
      }


int change_magnification ( void )
/*  This function changes the mag factor for the fields display */
      {
      int itemp;

      /* Loop until user inputs a good value */
      do    {

            /*    Tell the user what current mag factor is */
            sprintf ( ctbuf, "Current magnification is %d: Input a new value:
", mag );
            put_message ( ctbuf );

            /*  Now get the new value */
            itemp = wait_for_buffered_kbin ( 2 );  /* up to two digits */
            if ( itemp < 0 ) return 0;    /* abort */

            clear_message ();
            /* Now convert to an integer */
            itemp = atoi ( kbin_buf );
            if ( itemp <= 0 )
                  {
                  /* bad input, so tell him to try again */
                  sprintf ( ctbuf, "Unrecognizable integer string:%s , Please
try again", kbin_buf );
                  put_message_with_wait ( ctbuf );
                  }

            } while ( itemp <= 0 );

      /* Got a good mag factor so make it happen */
       undraw_field ( -1 );
       mag = itemp;
      fix_field_boxes ( -1, 1 );

      /* Fini */
      return 0;
      }


int reorder_fields ( void )
/*    This function will change the order of field presentation */
      {
      field_type f;
      int src, dest, i;

       /* Source will be the index of the field to move */
       if ( ( fselect < 0 ) || ( fselect >= nfields ) )
             {
             sprintf ( ctbuf, "In reorder fields with fselect = %d", fselect
);
             return 0;
```

```
            }
       else src = fselect;
      highlight_field ( src );

      /* Now have him select the destination for the move */
      do    {
            /* Put the destination message */
            dest = -1;
            put_message ( "Use left button to select the destination field" );
            /* and allow user to escape */
            if ( wait_for_user_event () ) return 0;

            /* See if mouse is in a field box */
            if ( buttons.left )
                  {
                  dest = ismouse_in_field_box ();
                  if ( dest == src )
                        {
                        put_message_with_wait ("You can't move a field onto
itself" );
                        dest = -1;
                    }
            else if ( dest < 0 )
                put_message_with_wait ("Identify a destination field box" );
                  }

            /* Loop until user gets it right */
            } while ( dest < 0 );

    /* First, save src data to the holding area */
    unhighlight_field ();
    undraw_field ( -1 );
    copy_field_data ( &fieldset[src], &f );

      /* Now loop, copying the fields data into its' new position */
      if ( src < dest )
            {
            /* src < dest.  Copy going up in array */
        for ( i=src; i<dest; i++ )
            copy_field_data ( &fieldset[i+1], &fieldset[i] );
            } /* End src<dest */

      else

            {
            /* src > dest.  Copy going down in array */
        for ( i=src; i>dest; i-- )
            copy_field_data ( &fieldset[i-1], &fieldset[i] );
            }     /* end else */

    /* Now copy buffer to destination */
    copy_field_data ( &f, &fieldset[dest] );

      /* field data is up-to-date.  change all field boxes for the new order
*/
       fix_field_boxes ( -1, 1 );

      /* Set fselect to point at the field previously selected */
```

```
      fselect = dest;
      highlight_field ( fselect );

      clear_message ();
      /* All done */
      return 0;
      }



int select_field ( void )
      {
      int temp;
      point_type p;

      if ( fselect < 0 )
            {
            do  {
                  /*
                   *  Mouse is not in a field box.  Position mouse in the
field
                   *    box and request user identify a field
                   */
                  p.x = field_box.pt1.x / 2 + field_box.pt2.x / 2;
                  p.y = field_box.pt1.y / 2 + field_box.pt2.y / 2;
                  set_mouse_position ( &p );

                  /* Now put the select message */
                  put_message ( "Place mouse in a field and hit left button"
);

                  /* Wait for user to do something */
                  if ( wait_for_user_event() ) return 0;
                  else if ( buttons.left )
                        fselect = ismouse_in_field_box ( );

                  /* Continue looping until it is in a field box */
                  } while ( fselect < 0 );

            }     /* End fselect test */

      /* Clear the message and highlight the selected box */
      clear_message();
      highlight_field ( fselect );

      /* Now wait til left button is released */
      while ( buttons.left );

      draw_frame_field ( fselect, 0xff, 10 );

      /* Now, enter the command loop */
      do   {

            /* If reentry for left button event, see if its for a new field */
            if ( buttons.left )
                  {
```

```
                    /* Is the mouse in a new field box */
                    temp = ismouse_in_field_box ( );
                    if ( ( temp >= 0 ) && ( temp != fselect ) )
                          {
                          /* A new box was identified */
                          undraw_frame_field ( fselect, 0xff, 10 );  /* undraw
old */
                          fselect = temp;
                          highlight_field ( fselect );
                          draw_frame_field ( fselect, 0xff, 10 );  /* draw new
*/
                          }
                    else
                          {
                          /* Mouse is still in selected field, yet user has hit
left
                                button again.  He may be selecting a field char
box */
                          temp = ismouse_in_fieldchar_box ( fselect );
                          if ( temp >= 0 )
                                {
                                /* A character was selected.  Initiate move_char
*/
                                cselect = temp;
                                highlight_field_char ( fselect, cselect );
                                undraw_frame_field ( fselect, 0xff, 10 );
                                move_char();
                                draw_frame_field ( fselect, 0xff, 10 );
                                }

                          }     /* End else */

                    }     /* End buttons.left test */

              /* Now, Set up and display commands */
              enable_command ( &csize );
              enable_command ( &nchars );
              enable_command ( &delete );
              if ( nfields > 1 )
                    enable_command ( &reorder );

              /* Now wait for user to do something */
              } while ( NOT wait_for_user_event() );

        /* Exiting fselect */
        unhighlight_field();
        undraw_frame_field ( fselect, 0xff, 10 );
        fselect = -1;

        /* Fini */
        return 0;
        }


int change_char_size ( void )
/*    This function changes the char width for the field currently selected.
*/
```

```
      {
      int itemp, modified, i;
      point_type * p;
      box_type * b;
      float fx, fbias;
      char buf[100];

      do    {

            /* Tell user to use up/down arrows */
            sprintf ( buf, "%s%d%s%d", "Up/Dn--taller/smaller, right/left--
wider/narrower, current size ",
                  fieldset[fselect].cwidth, " x ", fieldset[fselect].cheight
);
            put_message ( buf );

            /* See if arrow has been hit */
            modified = 1;           /* Default case */
            itemp = check_edit_arrows();

            /* Change width/height if arrow has been hit */
            if      ( itemp == UP_ARROW )       fieldset[fselect].cheight ++;
            else if ( itemp == DOWN_ARROW )     fieldset[fselect].cheight --;
            else if ( itemp == RIGHT_ARROW )    fieldset[fselect].cwidth ++;
            else if ( itemp == LEFT_ARROW )     fieldset[fselect].cwidth --;
            else modified = 0;

            /* if anything was modified, fix and redraw the field */
            if ( modified )
                  {
                  if ( fieldset[fselect].cwidth < 1 ) fieldset[fselect].cwidth
= 1;
                  if ( fieldset[fselect].cheight < 1 )
fieldset[fselect].cheight = 1;
                  undraw_frame_field ( fselect, 0xff, 10 );
                  unhighlight_field ( );
                  undraw_field ( fselect );

                  p = fieldset[fselect].points;
                  b = &fieldset[fselect].fsbox;
                  fbias = (float)( b->pt2.x - b->pt1.x + 1 ) / (float)
fieldset[fselect].nchars;
                  fx = (float) ( b->pt1.x - frame_box.pt1.x );
                  for ( i=0; i<fieldset[fselect].nchars; i++ )
                        {
                        p[i].x = ftos ( fx );
                        p[i].y = b->pt1.y - frame_box.pt1.y;
                        fx += fbias;
                        }

                  fix_field_boxes ( fselect, 1 );
                  highlight_field ( fselect );
                  draw_frame_field ( fselect, 0xff, 10 );
                  }

            /* Continue looping til user is happy */
            }     while ( NOT wait_for_user_event () );
```

```
      clear_message ();
      /* Fini */
      return 0;
      }


int change_number_chars ( void )
/*
 *    Use mouse or arrow keys to change number of characters in the
 *    selected field
 */
      {
       int itemp, modified, nchars, nchars_old;
       point_type * p;

       do  {
             /* Tell user what to do */
             put_message ( "Use up/down arrows to inc/dec the number of
chars" );

             /* Setup for tests and get arrow indicator */
             modified = 1;       /* default assumes modified */
             nchars = nchars_old = fieldset[fselect].nchars;
             itemp = check_edit_arrows();

             /* Now set number of characters accordingly */
             if      ( itemp == UP_ARROW )   nchars++;
             else if ( itemp == DOWN_ARROW ) nchars--;
             else modified = 0;  /* it wasn't modified */


           /* If number of chars was modified, make it happen */
           if ( modified )
                      {
                      if ( nchars < 1 ) nchars = 1;

                      /* undraw the field */
                      unhighlight_field ( );
                      undraw_field ( fselect );
                      undraw_frame_field ( fselect, 0xff, 10 );

                      /* Reallocate if necessary */
                      fieldset[fselect].nchars = nchars;
                      if ( fix_field_allocation ( &fieldset[fselect] ) )
return 0;


                      /* See if user added a new point */
                      if ( nchars > nchars_old )
                            {
                            /* Need to create a new point */
                            p = fieldset[fselect].points;
                            p[nchars-1].x = p[nchars-2].x +
fieldset[fselect].cwidth;
                            p[nchars-1].y = p[nchars-2].y;
```

```
                              }


                                   /* Show the user what he did */
                              fix_field_boxes ( fselect, 0 );
                              highlight_field ( fselect );
                              draw_field ( fselect );
                              draw_frame_field ( fselect, 0xff, 10 );

                              }      /* End if modified test */

                        }      while ( NOT wait_for_user_event () );

                  clear_message ();
                  /* All done */
                  return 0;
                  }



int move_char ( void )
      {
      point_type * p;
      int modified, itemp;

       /* Check cselect, just to make sure */
       if ( ( cselect < 0 ) || ( cselect >= fieldset[fselect].nchars ) )
              return 0;

       draw_frame_char ( fselect, cselect, 0xff, 10 );

       /* Now enter the move loop */
       do  {
              enable_command ( &grab );

              /* Tell the user what to do and initialize */
              put_message ("Use arrows to move the selected character");
              p = &fieldset[fselect].points[cselect];
              modified = 1;

              /* Now get arrows status and decode */
              itemp = check_edit_arrows ();
              if       ( itemp == UP_ARROW )        p->y--;
              else if  ( itemp == DOWN_ARROW )      p->y++;
              else if  ( itemp == LEFT_ARROW )      p->x--;
              else if  ( itemp == RIGHT_ARROW )     p->x++;
              else modified = 0;

              if ( modified )
              {
                    if ( p->y < 0 )
                          p->y = 0;

                    if ( p->y + fieldset[fselect].cheight > device.lines )
                          p->y = device.lines - fieldset[fselect].cheight;

                    if ( p->x < frame_box.pt1.x - MARGIN - 1 )
```

```
                        p->x = frame_box.pt1.x - MARGIN - 1;

                    if ( p->x > frame_box.pt2.x - MARGIN -
fieldset[fselect].cwidth + 2)
                        p->x = frame_box.pt2.x - MARGIN -
fieldset[fselect].cwidth + 2;

                    change_field_char ( fselect, cselect );
                }

                /* See if user selected a different character */
                if ( buttons.left )
                        {
                        /* See if it is another field */
                        itemp = ismouse_in_field_box ( );
                        if ( ( itemp >= 0 ) && ( itemp != fselect ) )
                        {
                        /* User has selected a field box, so do it */
                                undraw_frame_char ( fselect, cselect, 0xff, 10
);
                                fselect = itemp;
                                cselect = 0;
                                highlight_field_char ( fselect, cselect );
                                draw_frame_char ( fselect, cselect, 0xff, 10 );
                                }

                        /* See if its another character in the field */
                        itemp = ismouse_in_fieldchar_box ( fselect );
                        if ( ( itemp >= 0 ) && ( itemp != cselect ) )
                                {
                                /* New character selected */
                                undraw_frame_char ( fselect, cselect, 0xff, 10
);
                                cselect = itemp;
                                draw_frame_char ( fselect, cselect, 0xff, 10 );
                                highlight_field_char ( fselect, cselect );
                                }

                        }   /* End buttons.left test */

                } while ( NOT wait_for_user_event () );

        undraw_frame_char ( fselect, cselect, 0xff, 10 );
        cselect = -1;
        unhighlight_field_char();
        clear_message ( );
        /* All done */
        return 0;
        }



int delete_field ( void )
        {
        int i;

    /* Verify fselect is reasonable */
    if ( ( fselect < 0 ) || ( fselect >= nfields ) )
```

```
        {
        /* Invalid field selection */
        sprintf ( ctbuf, "In delete_field with fselect = %d", fselect );
        return 0;
        }

      /* verify user really wants to delete the field */
    if ( confirm ( "Confirm to delete the selected field ?" ) > 0 )
        {
              unhighlight_field();        /* get rid of highlight box */
              undraw_frame_field ( fselect, 0xff, 10 );
              undraw_field ( -1 );        /* undraw all fields */

              /* The answer was yes.  Make its' array go away */
              free_field_arrays ( &fieldset[fselect] );

            /* Now copy higher fields down one */
              for ( i=fselect; i<nfields-1; i++ )
                        copy_field_data ( &fieldset[i+1], &fieldset[i] );

              /* and make it happen */
              nfields--;                                    /* decrement field
counter     */
              fix_field_boxes ( -1, 1 );     /* fix and redraw fields
      */
              if ( nfields >= 0 )
                        fselect = 0;
              else
                        fselect = -1;
              highlight_field ( fselect ) ;
              draw_frame_field ( fselect, 0xff, 10 ) ;
            }

      /* All Done */
      return 0;
      }



short ftos ( float x )
/* Function converts a float to short with rounding */
      {
      short sign=1, x1;
      float tp;

      /* Capture the sign */
      if ( x < 0 ) sign = -1;

      /* Now perform rounding.  Note no overflow checking */
      x1 = (short) ( fabs ( x ) );
    tp = (float) fabs ( (double) x ) - ( (float) x1 );
      if ( tp > 0.5f ) x1++;

      /* And return with the sign preserved */
      return sign * x1;
      }
```

```
void copy_field_data ( field_type * src, field_type * dest )
    {
    dest->nchars            = src->nchars;
    dest->cwidth            = src->cwidth;
    dest->cheight           = src->cheight;
    dest->fsbox.pt1.x       = src->fsbox.pt1.x;
    dest->fsbox.pt1.y       = src->fsbox.pt1.y;
    dest->fsbox.pt2.x       = src->fsbox.pt2.x;
    dest->fsbox.pt2.y       = src->fsbox.pt2.y;
    dest->fdbox.pt1.x       = src->fdbox.pt1.x;
       dest->fdbox.pt1.y       = src->fdbox.pt1.y;
       dest->fdbox.pt2.x       = src->fdbox.pt2.x;
    dest->fdbox.pt2.y       = src->fdbox.pt2.y;
    dest->nchars_allocated = src->nchars_allocated;
    dest->points            = src->points;
    dest->sbox              = src->sbox;
    dest->dbox              = src->dbox;
    }


int allocate_field_arrays ( field_type * f )
/*
 *    This function allocates a block of memory for the field
 *  structures' point, sbox and dbox arrays, and initializes
 *  the pointers for these arrays.  If malloc returns an error,
 *  this function returns non-zero.
 */
    {
    int size, n;
    point_type * p;
    box_type * bs, *bd;

    n = f->nchars + 2;
    size = n * ( sizeof( point_type ) + 2 * sizeof ( box_type ) );

    /* Now get the block of memory */
    p = ( point_type * ) malloc ( size );
    if ( p == NULL )
        return -1;

    /* Now set the pointers up */
    bs = ( box_type * ) & ( p  [n] );
    bd = ( box_type * ) & ( bs [n] );

       /* Put the data into the array */
       f -> points = p;
    f -> sbox = bs;
    f -> dbox = bd;
    f -> nchars_allocated = n;

    /* All done */
    return 0;
    }


int fix_field_allocation ( field_type * f )
```

```
/*
 *  This function will check and fix the allocation for the point and
 *  box arrays for a specific field, pointed to by f.
 *  Return: 0           no error
 *          nonzero:    could not reallocate
 */
    {
    int alloc, n, i;
    point_type *pnew, *pold;

      /* Need to check if we have enough space allocated */
    alloc = f -> nchars_allocated;
    n = f -> nchars;
    if ( n > alloc )
        {
        /* save the old point array */
        pold = f -> points;

        /* need to reallocate the point and box array */
        if ( allocate_field_arrays( f ) )
            {
            /* Could not reallocate */
            put_message_with_wait ( "Error: Couldn't reallocate field " );
                        put_message_with_wait ( "Best save and exit field
editor" );
                        f->nchars = n-1;
            return -1;
            }

        /* Initialize the new point array */
        pnew = f->points;
        for ( i=0; i<n-1; i++ )
            {
            pnew[i].x = pold[i].x;
            pnew[i].y = pold[i].y;
            }

        /* Kill the old point array */
        free ( pold );
        }

    /* Fini */
    return 0;
    }




void free_field_arrays ( field_type * f )
/*
 *  This function kills the field arrays, and zeros the pointers
 */
    {
    free ( f->points );
    f->points = NULL;
    f->sbox = NULL;
    f->dbox = NULL;
```

```
    f->nchars_allocated = 0;
       }


/*************************************************************************
 *
 *    FILES.C
 *
 *    This file contains the functions required to get and save the
 *  field, exemplar and training set data required by the body of
 *  the application.
 *
 *************************************************************************
 */

#include <malloc.h>
#include <stdio.h>
#include <string.h>

#include "net.h"
#include "network.h"

/* Declare locals */
static FILE * f;

/* Declare new functions */
int new_fieldset ( char * );
int new_exempset ( char *, char * );
int new_trainset ( char *, char * );

int get_field_set ( void )
      {
      int i, temp;
      short nchars, j, x, y, width, height;

      /* Open the file for readonly */
      if ( ( f = fopen ( filfield, "r" ) ) == NULL )
            {
            sprintf ( ctbuf, " %s : file does not exist", filfield );
            return new_fieldset( ctbuf );
            }

      /* The file does exist.  Get the number of fields */
      if ( fscanf( f,"%d\n", &nfields ) != 1)
            {
            fclose (f);
            return new_fieldset( "Warn: nfields not valid" );
            }

      /* Allocate memory for the field data, with some space for new ones */
      nfields_allocated = nfields + MAX_FIELDS;
      temp = nfields_allocated * sizeof(field_type);
      fieldset = (field_type *) malloc ( temp );
      if ( fieldset == NULL )
            {
            fclose ( f );
            return new_fieldset( "Error: allocating fieldset" );
```

```
            }

      /* Time to fill the fields data */
      for (i = 0; i< nfields ; i++)
            {
            if ( fscanf( f, "%d%d%d", &nchars, &width, &height ) != 3 )
                  {
                  nfields = i-1;
                  sprintf ( ctbuf, "Warn: header for field %d invalid", i );
                  fclose ( f );
                  return new_fieldset ( ctbuf );
                  }

            /* Got the field header so fill it in */
            fieldset[i].nchars = nchars;
            fieldset[i].cwidth = width;
            fieldset[i].cheight = height;

            /* Now allocate memory for the point and box arrays */
            if ( allocate_field_arrays ( &fieldset[i] ) )
                  {
                  /* an error was returned */
                  nfields = i-1;
                  sprintf ( ctbuf, "Error: allocating arrays for field %d", i
);
                  fclose ( f );
                  return new_fieldset ( ctbuf );
                  }

            /* Now read the point data for each character */
            for ( j=0; j<nchars; j++ )
                  {
                  /* Read the data */
                  if ( fscanf( f, "%d%d", &x, &y ) != 2 )
                        {
                        /* Error condition */
                        nfields = i;
                        sprintf ( ctbuf, "Warn: field %d, char %d invalid", i,
j );
                        fclose ( f );
                        return new_fieldset ( ctbuf );
                        }
                        fieldset[i].points[j].x = x;
                  fieldset[i].points[j].y = y;

                  }     /* End character coordinate reading loop */

            }     /*    End field read loop */

      /* We be done */
      fclose ( f );
      return 0;
      }


int new_fieldset ( char * msg )
/* This function kills any existing fields read, then allocates for new */
      {
```

```
        int temp;

        /* Put the message with a pause for user acknowledgement */
        put_message_with_wait ( msg );


        /* Free up anything already allocated */
        if ( nfields > 0 ) kill_field_set ();
        else if ( fieldset != NULL ) free (fieldset);

        /* Now allocate space for a bunch of fields */
        temp = MAX_FIELDS * sizeof ( field_type );
        fieldset = ( field_type * ) malloc ( temp );
        if ( fieldset == NULL ) return -1;

        /* Set the global fieldset descriptors */
        nfields_allocated = MAX_FIELDS;
        nfields = 0;

        /* And get out of Dodge */
        return 0;
        }



int save_field_set ( void )
        {
        short i, width, height, x, y;
        int j, nchars;

        /* Save the field data */
        if ( nfields <= 0 ) return 1;  /* nothing to save */

        /*
         *  Open the field file for write-only.  If the file exists,
         *  it will be trashed
         */
        f = fopen ( filfield, "w" );
        if ( f == NULL )
              {
              printf ( "Error opening %s for output\n", filfield );
              return -1;
              }

        /*  Now write the header, indicating the number of fields */
        fprintf(f, "%d\n\n", nfields);
        for (i=0; i < nfields; i++)
              {
              /* Write the field specific data */
              nchars = fieldset[i].nchars;
              width = fieldset[i].cwidth;
              height = fieldset[i].cheight;
              fprintf(f, "    %d %d %d\n", nchars, width, height );

              /* Now write the character positions for the field */
              for ( j=0; j<fieldset[i].nchars; j++ )
                    {
                    x = fieldset[i].points[j].x;
```

```
            y = fieldset[i].points[j].y;
                  fprintf(f, "        %d    %d\n", x, y);
                  }
            }   /* end field write loop */

      /* All done */
      fclose ( f );
      return 0;
      }


void kill_field_set ( void )
      {
      int i;

      /* free the point array for each field defined */
      for ( i=0; i<nfields; i++ )
            free_field_arrays ( &fieldset[i] );

      /* Now free the fields array */
      free ( fieldset );
      fieldset = NULL;
      nfields_allocated = nfields = 0;

      return;
      }


int get_exemplar_set ( char *filename )
/*
 *    This function reads the exemplar data file, whose name is pointed
 *    to by f.  There are two files of this type, the raw exemplar data
 *    file and the normalized exemplar data file.  Hence, f will always
 *    point at one or the other.
 */
      {
      int v, i, width, height, temp, ndots;
      unsigned int utemp;
      BYTE * cptr;

      /* See if the exemplar set file exists.  If so, open it */
      f = fopen ( filename, "r" );
      if ( f == NULL )
            {
            sprintf ( ctbuf, "Warn: cannot open file %s", filename );
            return new_exempset( ctbuf, filename );
            }

      /*  File exists.  Read in the # of exemplar chars in the file */
      if ( fscanf ( f, "%d", &(nexemp) ) != 1 )
            {
            fclose (f);
            return new_exempset( "Warn: number of exemps not valid", filename
);
            }
```

```
      /* Now allocate memory for the exemplar set */
      nexemp_allocated = nexemp + MAX_EXEMPLARS;
      temp = nexemp_allocated * sizeof ( exemp_type );
      exempset = (exemp_type *) malloc ( temp );
      if ( exempset == NULL )
            {
            fclose (f);
            return new_exempset( "Error: could not allocate exempset",
filename );
            }

      /* Now read in the input vectors for training */
      for (v=0; v<nexemp; v++)
            {
            /* Get the characters' header data */
            if ( fscanf (f, "%x%d%d", &utemp, &width, &height ) != 3 )
                  {
                  nexemp = v-1;
                  sprintf ( ctbuf, "Warn: could not read exemp char header for
%d", v);
                  fclose (f);
                  return new_exempset ( ctbuf, filename );
                  }
            exempset[v].ch = ( char ) utemp;
            exempset[v].cwidth = width;
            exempset[v].cheight= height;

            /*  Got character and size.  Allocate memory for the dot matrix */
            ndots = exempset[v].cwidth * exempset[v].cheight;
            cptr  =  ( BYTE * ) malloc ( ndots * sizeof ( BYTE ) );
            if ( cptr == NULL )
                  {
                  nexemp = v-1;
                  sprintf ( ctbuf, "Warn: could not allocate space for exemp
%d", v );
                  fclose (f);
                  return new_exempset ( ctbuf, filename);
                  }

            /* Ready to fill the dot matrix */
            exempset[v].dm_ptr = cptr;
            for(i=0; i<ndots; ++i)
                  {
                  if ( ( fscanf(f, "%x", &utemp ) ) != 1 )
                        {
                        /* Got an error during read.  Kill the exemplar set */
                        nexemp = v;
                        sprintf ( ctbuf, "Warn: Invalid exemp char data for
char %d, dot %d", v, i);
                        fclose (f);
                        return new_exempset ( ctbuf, filename );
                        }
                  *cptr++ = ( BYTE ) utemp;
                  }   /* end dot matrix loop */
            }   /* end exemplar set loop */

      /* All done */
      fclose(f);
```

```
        return 0;
        }


int new_exempset ( char * msg , char *filename)
/* This function destroys exemplar data read, then allocates for new */
        {
        int temp;

        /* Put the message with a pause for user acknowledgement */
        put_message_with_wait ( msg );

        remove ( filename );

        /* Free up anything already allocated */
        if ( nexemp > 0 ) kill_exemplar_set ( );
        else if ( exempset != NULL ) free ( exempset );

        /* Now allocate space for a bunch of exemplar characters */
        temp = MAX_EXEMPLARS * sizeof ( exemp_type );
        exempset = ( exemp_type * ) malloc ( temp );
        if ( exempset == NULL ) return -1;

        /* Set the global fieldset descriptors */
        nexemp_allocated = MAX_EXEMPLARS;
        nexemp = 0;

        /* And get out of Dodge */
        return 0;
        }


int save_exemplar_set ( char *filename )
        {
        short i, j, k;
        unsigned int utemp;
        BYTE * ptr;

        if ( nexemp <= 0 ) return 0;     /* Don't need to do anything */

        /* Open the file.  If the file exists, it will be trashed */
        if ( ( f = fopen ( filename, "w" ) ) == NULL )
                {
                printf ( "Error opening %s for output\n", filename );
                return -1;
                }

        /* Write the number of exemplars */
        fprintf(f, "%d\n", nexemp );

        /*
         *    Now loop through each exemplar character, printing the size data
         *  and dot matrix
         */
        for ( i=0; i<nexemp; i++ )
                {
```

```
            utemp = (unsigned) exempset[i].ch;
            fprintf (f, "%4x  %4d  %4d\n", utemp, exempset[i].cwidth,
                              exempset[i].cheight );
            ptr = exempset[i].dm_ptr;
            for ( k=0; k<exempset[i].cheight; k++ )
                  {
                  for ( j=0; j<exempset[i].cwidth; j++ )
                        fprintf ( f, "  %2x", *ptr++ );
                  fprintf ( f, "\n" );
                  }
            }

      /* All done */
      fclose ( f );
      return 0;
      }


void kill_exemplar_set ( void )
      {
      int i;

      /* free the exemplar set dot matrix pointers */
      for ( i=0; i<nexemp; i++ )
            free ( exempset[i].dm_ptr );

      /* Now free the exemplar set array */
      free ( exempset );
      exempset = NULL;
      nexemp_allocated = nexemp = 0;

      return;
      }




int get_train_set ( char *filename )
/*
 *    This function reads the train    data file, whose name is pointed
 *    to by f.  There are two files of this type, the raw train data
 *    file and the normalized train data file.  Hence, f will always
 *    point at one or the other.
 */
      {
      int v, i, temp, ndots;
      unsigned int utemp;
      BYTE * cptr;

      /* See if the train    set file exists.  If so, open it */
      f = fopen ( filename, "r" );
      if ( f == NULL )
            {
            sprintf ( ctbuf, "Warn: cannot open file %s", filename );
            return new_trainset( ctbuf, filename );
            }
```

```
      /*  File exists.  Read in the # of train chars in the file */
      if ( fscanf ( f, "%d %d", &(ntrain), &K ) != 2 )
            {
            fclose (f);
            return new_trainset( "Warn: number of trains not valid", filename
);
            }

      /* Now allocate memory for the train set */
      ntrain_allocated = ntrain + MAX_EXEMPLARS;
      temp = ntrain_allocated * sizeof ( train_type );
      trainset = (train_type *) malloc ( temp );
      if ( trainset == NULL )
            {
            fclose (f);
            return new_trainset( "Error: could not allocate trainset",
filename );
            }


      if ( fscanf (f, "%d%d", &trwidth, &trheight ) != 2 )
            {
            sprintf ( ctbuf, "Warn: could not read train header");
            fclose (f);
            return new_trainset ( ctbuf, filename );
            }

      I = trwidth * trheight;


      /* Now read in the input vectors for training */
      for (v=0; v<ntrain; v++)
            {
            /* Get the characters' header data */
            if ( fscanf (f, "%x%d", &utemp, &trainset[v].out_index ) != 2 )
                  {
                  ntrain = v-1;
                  sprintf ( ctbuf, "Warn: could not read train char header for
%d", v);
                  fclose (f);
                  return new_trainset ( ctbuf, filename );
                  }
            trainset[v].ch = ( char ) utemp;

            /*  Got character and size.  Allocate memory for the dot matrix */
            ndots = trwidth * trheight;
            cptr  =  ( BYTE * ) malloc ( ndots * sizeof ( BYTE ) );
            if ( cptr == NULL )
                  {
                  ntrain = v-1;
                  sprintf ( ctbuf, "Warn: could not allocate space for train
%d", v );
                  fclose (f);
                  return new_trainset ( ctbuf, filename);
                  }

            /* Ready to fill the dot matrix */
            trainset[v].dm_ptr = cptr;
```

```
            for(i=0; i<ndots; ++i)
                  {
                  if ( ( fscanf(f, "%x", &utemp ) ) != 1 )
                        {
                        /* Got an error during read.  Kill the train set */
                        ntrain = v;
                        sprintf ( ctbuf, "Warn: Invalid train char data for
char %d, dot %d", v, i);
                        fclose (f);
                        return new_trainset ( ctbuf, filename );
                        }
                  *cptr++ = ( BYTE ) utemp;
                  }   /* end dot matrix loop */
            }   /* end train    set loop */

      /* All done */
      fclose(f);
      return 0;
      }


int new_trainset ( char * msg , char *filename)
/* This function destroys train    data read, then allocates for new */
      {
      int temp;

      /* Put the message with a pause for user acknowledgement */
      put_message_with_wait ( msg );

      remove ( filename );

      /* a new trainset w/ potential diff char sizes invalidates current net.
            ditch it  !!!! */
      remove ( filnet );

      /* Free up anything already allocated */
      if ( ntrain > 0 ) kill_train_set ( );
      else if ( trainset != NULL ) free ( trainset );

      /* Now allocate space for a bunch of trainlar characters */
      temp = MAX_EXEMPLARS * sizeof ( train_type );
      trainset = ( train_type * ) malloc ( temp );
      if ( trainset == NULL ) return -1;

      /* Set the global fieldset descriptors */
      ntrain_allocated = MAX_EXEMPLARS;
      ntrain = 0;

      /* And get out of Dodge */
      return 0;
      }


int save_exemp_as_train ( char *filename )
      {
      short i, j, k, bigk;
      unsigned int utemp;
```

```
        BYTE * ptr;

        if ( nexemp <= 0 ) return 0;     /* Don't need to do anything */

        /* Open the file.  If the file exists, it will be trashed */
        if ( ( f = fopen ( filename, "w" ) ) == NULL )
              {
              printf ( "Error opening %s for output\n", filename );
              return -1;
              }

        bigk = 1;
        for ( i=1; i < nexemp; i++ )
          if (exempset[i].ch != exempset[i-1].ch)
               bigk++;

        /* Write the number of train */
        fprintf(f, "%d %d\n", nexemp, bigk );
        fprintf (f, "%4d  %4d\n",  exempset[0].cwidth,  exempset[0].cheight );

        /*
         *    Now loop through each training character, printing the size data
         *  and dot matrix
         */
        for ( i=0; i<nexemp; i++ )
              {
              utemp = (unsigned) exempset[i].ch;
              fprintf (f, "%4x  %4d\n", utemp, 0 );
              ptr = exempset[i].dm_ptr;
              for ( k=0; k<exempset[0].cheight; k++ )
                    {
                    for ( j=0; j<exempset[0].cwidth; j++ )
                          fprintf ( f, "  %2x", *ptr++ );
                    fprintf ( f, "\n" );
                    }
              }


        /* a new trainset w/ potential diff char sizes invalidates current net.
              ditch it  !!!! */
        remove ( filnet );


        /* All done */
        fclose ( f );
        return 0;
        }



void kill_train_set ( void )
        {
        int i;

        /* free the train  set dot matrix pointers */
        for ( i=0; i<ntrain; i++ )
              free ( trainset[i].dm_ptr );
```

```
      /* Now free the train set array */
      free ( trainset );
      trainset = NULL;
      ntrain_allocated = ntrain = 0;

      return;
      }




int fexist ( char * filename )
/*
 *    This function determines if a file exists by opening the file.
 *  If the file exists, it is closed and function returns nonzero (TRUE).
 *    Otherwise, return zero (FALSE).
 */
      {

      f = fopen ( filename, "r" );
      if ( f == NULL )
            return 0;
      else
            fclose ( f );
      return 1;
      }


/*****************************************************************************
 *
 *    FRAME.C
 *
 *    This file contains the application for customizing the video
 *    characteristics associated with a given source.  One can lighten
 *    or darken, resize, etc.  Editing the frame is always possible
 *    within the application, but once fields and exemplar characters
 *    are defined, be advised that changing frame timing or size
 *    will really screw up all of the downstream applications.
 *
 *****************************************************************************
 */

#include <conio.h>
#include <stdlib.h>

#include "mouse.h"
#include "net.h"
#include "ntiga.h"
#include "vcrsmfg.h"


#define SIZE_CBUF 27

/* Declare globals */
box_type frame_box;           /* Video frames will go here */
```

```
/* Declare locals to this module */
static box_type options_box;     /* Where the device parameters will be placed
*/
static box_type arrows_box;          /* Defines where to put editing arrows */
static box_type devbox[10];      /* This is for boxing the selected device
parameter */
static point_type devpts[10];    /* This is for outputing the device parameter
description */
static char cbuf[SIZE_CBUF];     /* Buffer for the device description */
static int select = -1;          /* Currently selected device parameter */

/* Declare function prototypes */
void init_frame_editor ( void );
void change_command ( int );
void change_white ( int );
void change_black ( int );
void change_hsize ( int );
void change_vsize ( int );
void change_hhold ( int );
void change_vhold ( int );
void change_clock ( int );
void change_dclock ( int );
int frame_grab ( void );
int exit_frame_editor ( void );


static command_type grab      =
      {     "GrabFrame",
            frame_grab,
            "Grab a new video frame" };

/*
 *  Declare the structure used for displaying device titles
 *  The structure contains the title, and a pointer to the function
 *  used to change the particular device data element.  Then fill
 *  fill it in.
 */
struct dev_struc
       {
                  char * title;
             void ( * change_func )( int );
       } dev_data[] =  {
                                          { "Command Type       = ",
change_command },
                                            { "White Level        = ",
change_white },
                                           { "Black Level        = ",
change_black },
                                          { "Horizontal Size    = ",
change_hsize },
                                            { "Vertical Size      = ",
change_vsize },
                                           { "Horizontal Holdoff = ",
change_hhold },
                                          { "Vertical Holdoff   = ",
change_vhold },
```

```
                                        { "Sampling Clock      = ",
change_clock },
                                        { "Delay Clock         = ",
change_dclock }
                                    };
int num_dev = sizeof ( dev_data ) / sizeof ( struct dev_struc );


int frame_editor ( void )
      {
      int itemp;

      /* Initialize */
      clear_display();
      init_frame_editor();


      /* Now loop until user hits escape */
      do {
            enable_command ( &grab );
            if ( buttons.left )
                  {

                  /* See if mouse is in increase/decrease boxes */
                  itemp = check_edit_arrows ( );
                  if ( ( itemp >= 0 ) && ( select > 0 ) )
                        {
                        /* If up or down, call the function indirectly */
                        if ( itemp == UP_ARROW )
                              ( * ((dev_data[select]).change_func) ) ( -1 );
                        else if ( itemp == DOWN_ARROW )
                              ( * ((dev_data[select]).change_func) ) ( 1 );
            frame_grab ();
                        }

                  /* It wasn't an arrow box.  See if user selected another
param */
                  itemp = ismouse_in_box_array ( devbox, num_dev );
                  if ( itemp >= 0 )
                        {
                        /* Another parameter.  Undraw box for current select
*/
                        undraw_box ( &devbox[select] );

                        /* Now indicate new selection with a new box */
                        select = itemp;
                        draw_box ( &devbox[select] );

                        /* If it is a change command, call that function here
*/
                        if ( select == 0 )
                              ( * ((dev_data[select]).change_func) ) ( 1 );

                        }     /* End new param test */

                  }     /* End mouse left button test */
```

```
            /* Now get another frame */
            clear_message ();

            } while ( NOT wait_for_user_event() );

      exit_frame_editor ( ) ;

      /* All done, so clear display and exit */
      clear_display();
      return 0;
      }


void init_frame_editor ( void )
      /* This function initializes the global parameters used thoughout the
         frame editor processing */
      {
      int i, x, y, box_spacing;


      /* Now set up the box where the device options will go */
      options_box.pt1.x = frame_box.pt2.x + MARGIN;
      options_box.pt1.y = frame_box.pt1.y;
      options_box.pt2.x = display_box.pt2.x;
      options_box.pt2.y = frame_box.pt2.y;

      /* Now set up the box where the editing arrows will go */
      arrows_box.pt1.x = options_box.pt1.x;
      arrows_box.pt1.y = options_box.pt2.y + MARGIN + font.charhigh;
      arrows_box.pt2.x = options_box.pt2.x;
      arrows_box.pt2.y = display_box.pt2.y - MARGIN;

      /* Now compute the device parameter display box data */
      x = ( options_box.pt1.x + options_box.pt2.x ) / 2
            - SIZE_CBUF * font.charwide / 2 + FONT_SPACING;
      box_spacing = font.charhigh + 2 * FONT_SPACING;
      y = ( options_box.pt1.y + options_box.pt2.y ) / 2
            - num_dev * box_spacing / 2;

      /* Loop until all device boxes are defined */
      for ( i=0; i<num_dev; i++ )
            {
            devbox[i].pt1.x = x;
            devbox[i].pt1.y = y;
            devbox[i].pt2.x = x + SIZE_CBUF * font.charwide + 2 *
FONT_SPACING;
            devbox[i].pt2.y = y + box_spacing - 1;
            y += box_spacing;

            /* Now set up the point where the device parameter text will be
put */
            devpts[i].x = devbox[i].pt1.x + FONT_SPACING;
            devpts[i].y = devbox[i].pt1.y + FONT_SPACING;

            /* Now execute the function to write the device data on the
display */
            select = i;
              ( * ((dev_data[i]).change_func) ) ( 0 );
```

```
        }

    /* Draw the major field boxes and place titles */
    put_screen_title ( "FRAME EDITOR" );
    put_box_title ( "CURRENT FRAME", &frame_box );
    put_box_title ( "DEVICE FILE CONFIGURATION", &options_box );

    /* Now draw the arrows in the arrows box */
    init_arrows_edit_box ( &arrows_box );

    /* Indicate to user we are getting a frame, then get it */
    put_message ( "Grabbing a frame" );
    get_frame (  );
    put_message ( "" );

    /* Fini */
    }


int exit_frame_editor ( void )
    {
    int itemp;

    /* Save the device data as user directs */
    if ( fexist ( filframe ) )
        {
        itemp = confirm ( "Replace the modified frame data file? " );
        if ( itemp < 0 )  return 0;
        else if ( itemp > 0 ) save_device_data ( filframe );
        }

    /* The file doesn't exist so save the data */
    else save_device_data ( filframe );

    /* Indicate we be done */
    return 1;
    }

int frame_grab ( void )
    {
    box_type box;

    /* Initialize the box for the border */
    box.pt1.x = frame_box.pt1.x-1;
    box.pt1.y = frame_box.pt1.y-1;
    box.pt2.x = frame_box.pt1.x + device.pixels + 1;
    box.pt2.y = frame_box.pt1.y + device.lines + 1;

    /* Clear the current frame box */
    clear_box ( &frame_box );

    /* get frame returns non-zero if a problem occurred */
    get_frame (  );

    /* Redraw frame box border */
    draw_box ( &box );
```

```
        /* Fini */
        return 0;
        }


void change_command ( int ccmd )
/*
 *  This function changes the discrete word command parameter in the
 *  device data structure.  this is the only function which requires
 *  keyboard input.  If ccmd is 1, the function expects the user to
 *  change the command.  If it is anything else, the function simply
 *  outputs the parameter string to the options box.
 */
        {
        unsigned int ok;
        char * ctempptr;

        if ( ccmd == 1 )
              {
              put_message ( "Input the new command value:  " );
              wait_for_buffered_kbin ( 4 );  /* 4 characters expected in hex */

              /* The user input is in kbin_buf.  convert to an unsigned int */
              ok = (unsigned int) strtoul ( kbin_buf, &ctempptr, 16 );

              /* if the conversion took, ok is nonzero */
              if ( ok )
                    {
                    device.command = ok;
                    set_clock ( );
                    }
              else
                    put_message ( "Invalid command format--no change effected"
);
              }

        /* Regardless of what happened, put the string in the options box */
        sprintf ( cbuf, "%s%5x", dev_data[select].title, device.command );
        out_text ( devpts[select], cbuf, SIZE_CBUF, NORMAL_TEXT );
        put_message ( "" );

        /* Fini */
        return;
        }



void change_white ( int ccmd )
/*      This function changes the white or brightness level */
        {

        /* increment or decrement the current white level */
        if      ( ccmd < 0 ) device.white -= 0.05f;
        else if ( ccmd > 0 ) device.white += 0.05f;

        /* Now limit it to allowable range */
        if ( device.white > LMAX ) device.white = LMAX;
        if ( device.white < LMIN ) device.white = LMIN;
```

```
      /* Make it happen */
      set_levels (  );

      /* Now create and output the string to the options box */
      sprintf ( cbuf, "%s%5.1f", dev_data[select].title, device.white );
      out_text ( devpts[select], cbuf, SIZE_CBUF, NORMAL_TEXT );

      /* Fini */
      }


void change_black ( int ccmd )
/*    This function changes the black level, effecting overall contrast */
      {

      /* Increment or decrement the current level */
       if      ( ccmd < 0 ) device.black += 0.05f;
       else if ( ccmd > 0 ) device.black -= 0.05f;

      /* Now limit it to the allowable range */
      if ( device.black > LMAX ) device.black = LMAX;
      if ( device.black < LMIN ) device.black = LMIN;

      /* and make it happen */
      set_levels (  );

      /* Format and output the string to the options box */
      sprintf ( cbuf, "%s%5.1f", dev_data[select].title, device.black );
      out_text ( devpts[select], cbuf, SIZE_CBUF, NORMAL_TEXT );

      /* Fini */
      }


void change_hsize( int ccmd )
/* This function changes the horizontal size of the frame */
      {

      /* First, increment or decrement current size */
      if ( ( ccmd < 0 ) && ( device.pixels < display_size.x ) )
            device.pixels ++;
      else if ( ( ccmd > 0 ) && ( device.pixels > 0 ) )
            device.pixels --;

       /* Now modify the box the frame goes in */
      undraw_box_with_margin ( &frame_box, 1 );
      frame_box.pt2.x = frame_box.pt1.x + device.pixels - 1;
      frame_box.pt2.y = frame_box.pt1.y + device.lines - 1;
      draw_box_with_margin ( &frame_box, 1 );

      /* Now call the function that makes it happen */
      set_hvsize ();

      /* Format and output the string to the options box */
      sprintf ( cbuf, "%s%5d", dev_data[select].title, device.pixels );
      out_text ( devpts[select], cbuf, SIZE_CBUF, NORMAL_TEXT );
```

```
        /* Fini */
        }


void change_vsize( int ccmd )
/*      This function changes the vertical size of the frame */
        {

        /* Increment or decrement the vertical size */
        if ( ( ccmd < 0 ) && ( device.lines < display_size.y ) )
              device.lines ++;
        else if ( ( ccmd > 0 ) && ( device.lines > 0 ) )
              device.lines --;

        /* Now modify the box the frame goes in */
        undraw_box_with_margin ( &frame_box, 1 );
        frame_box.pt2.x = frame_box.pt1.x + device.pixels - 1;
        frame_box.pt2.y = frame_box.pt1.y + device.lines - 1;
        draw_box_with_margin ( &frame_box, 1 );

        /* Now make it happen */
        set_hvsize ();

        /* Format and output the string to the options box */
        sprintf ( cbuf, "%s%5d", dev_data[select].title, device.lines );
        out_text ( devpts[select], cbuf, SIZE_CBUF, NORMAL_TEXT );

        /* Fini */
        }


void change_hhold( int ccmd )
/* This function changes the horizontal holdoff */
        {
        unsigned char hhold;

        /* Get the current holdoff, in the most significant byte */
        hhold = (unsigned char ) ( ( device.holdoffs & 0xFF00 ) >> 8 );

        /* Now increment/decrement based on the incoming command */
        if      ( ( ccmd < 0 ) && ( hhold < 0xFF ) ) hhold++;
        else if ( ( ccmd > 0 ) && ( hhold > 0x00 ) ) hhold--;

        /* Set the new command back into the device structure and output it */
        device.holdoffs = ( device.holdoffs & 0xFF ) | ( hhold << 8 );
        set_holdoffs ();

        /* Format and output the string to the options box */
        sprintf ( cbuf, "%s%5x", dev_data[select].title, hhold );
        out_text ( devpts[select], cbuf, SIZE_CBUF, NORMAL_TEXT );

        /* Fini */
        }


void change_vhold( int ccmd )
/*      This function changes vertical holdoff */
        {
```

```
    unsigned char vhold;

    /* Get the current value for holdoff, in the least significant byte */
    vhold = ( unsigned char ) ( device.holdoffs & 0xFF );

    /* Now increment/decrement based on the incoming command */
     if      ( ( ccmd < 0 ) && ( vhold < 0xFF ) ) vhold++;
     else if ( ( ccmd > 0 ) && ( vhold > 0x00 ) ) vhold--;

    /* Put it back in the device structure and output */
    device.holdoffs = ( device.holdoffs & 0xFF00 ) | ( vhold );
    set_holdoffs ();

    /* Format and output the string to the options box */
    sprintf ( cbuf, "%s%5x", dev_data[select].title, vhold );
    out_text ( devpts[select], cbuf, SIZE_CBUF, NORMAL_TEXT );

    /* Fini */
    }


void change_clock( int ccmd )
    {
    float max0, max1, min0, min1, inc, idiv= 0.0f, utemp;

    utemp = device.command & 0x30;
    if ( utemp == 0 ) idiv = 1.0f;
    else if ( utemp == 0x10 ) idiv = 2.0f;
    else if ( utemp == 0x30 ) idiv = 4.0f;
    max1 = CMAX1 / idiv;
    min1 = CMIN1 / idiv;
    max0 = CMAX0 / idiv;
    min0 = CMIN0 / idiv;
    inc = ( max0 - min0 ) / 256;

    utemp = device.clock/idiv;
    if      ( ccmd < 0 )
          {
          utemp += inc;
          if ( ( utemp > max0 ) && ( utemp < min1 ) ) utemp = min1;
          }
    else if ( ccmd > 0 )
          {
          utemp -= inc;
          if ( ( utemp > max0 ) && ( utemp < min1 ) ) utemp = max0;
          }
    if ( utemp > max1 ) utemp = max1;
    if ( utemp < min0 ) utemp = min0;
    device.clock = utemp * idiv;
    set_clock (  );
    sprintf ( cbuf, "%s%5.2f", dev_data[select].title, utemp );
    out_text ( devpts[select], cbuf, SIZE_CBUF, NORMAL_TEXT );

    }


void change_dclock ( int ccmd )
    {
```

```
        float inc;

        inc = ( float ) ( ( CDMAX - CDMIN ) / 256.0f );
        if      ( ccmd < 0 ) device.dclk += inc;
        else if ( ccmd > 0 ) device.dclk -= inc;
        if      ( device.dclk > CDMAX ) device.dclk = CDMAX;
        else if ( device.dclk < CDMIN ) device.dclk = CDMIN;
        set_dclock ( );
        sprintf ( cbuf, "%s%5.1f", dev_data[select].title, device.dclk );
        out_text ( devpts[select], cbuf, SIZE_CBUF, NORMAL_TEXT );

        }

/*************************************************************************

   Name: mfg.c

   Description:  This file contains initialization and exit code for the
                                ITI MFG CLR card.

*************************************************************************/
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <direct.h>
#include <time.h>
#include <malloc.h>

#include <amc1.h>
#include <iff.h>

#include "vcrsmfg.h"
#include "ntiga.h"
#include "net.h"


#include <mfghost.h>

struct DEVICE device;

box_type frame_box;


static int src_aoi, dest_aoi, dm_aoi;

static int frame_aoi, save_aoi, t_aoi, s_aoi, b_aoi;


int initialize_mfg (void )
      {
      int i;

       device.pixels = 512;
       device.lines = 480;

       if (mfg_loadcnf("") != NO_ERROR)
       {
              printf("Unable to load MFG configuration file\n");
```

```
            return 1;
        }

        if (mfg_init() != NO_ERROR)
        {
            printf("Unable to initialize MFG\n");
            return 1;
        }


       i = amc1_sel_port(2);
       i = mfg_setvframe(GO);
       i = mfg_setgframe(GO);
       i = mfg_dacmode(PSEUDO_8_G);

       if (i == -1) return (1);


       src_aoi = mfg_gaoi_fbcreate(GO,0,0,50, 50);

       dest_aoi = mfg_gaoi_fbcreate(GO,0, 0, 50, 50);

       dm_aoi = mfg_gaoi_fbcreate(G,0, 0, 50, 50);

       frame_aoi = mfg_gaoi_fbcreate(G,frame_box.pt1.x, frame_box.pt1.y, 512,
480);


       save_aoi = mfg_gaoi_fbcreate(G, 0, 0, 1023,1023);
       b_aoi = mfg_gaoi_fbcreate(B, 0, 0, 1023,1023);
       t_aoi = mfg_gaoi_fbcreate(GO,0, 0, 1023,1023);
       s_aoi = mfg_gaoi_fbcreate(GB,0, 0, 1023,1023);

       /* Fini */
       return (0);
       }



void zoom_box ( box_type * sbox, box_type * dbox )
       {
       int i;


       i = mfg_gaoi_setarea(src_aoi, sbox->pt1.x, sbox->pt1.y,
            sbox->pt2.x - sbox->pt1.x + 1, sbox->pt2.y - sbox->pt1.y + 1);

       i = mfg_gaoi_setarea(dest_aoi, dbox->pt1.x, dbox->pt1.y,
            dbox->pt2.x - dbox->pt1.x + 1, dbox->pt2.y - dbox->pt1.y + 1);

       i += mfg_repzoom(src_aoi, dest_aoi, mag, mag);


       }
```

```
int get_frame( void )
      {
      int i;

      i = mfg_gaoi_setarea(frame_aoi, frame_box.pt1.x, frame_box.pt1.y,
            512, 480);

      i += mfg_snap(CAMERA, frame_aoi);


      return 0; /* rth 3/29/94 */

      }



void clear_framebuffer (void)
{
       mfg_wipe(0);
}


void clear_framebox (box_type *box)
{
       mfg_block (0,box->pt1.x, box->pt1.y,
            box->pt2.x - box->pt1.x, box->pt2.y - box->pt1.y, 0);
}



void copy_box ( box_type * sbox, box_type * dbox )
      {
      int i;

      i = mfg_gaoi_setarea(src_aoi, sbox->pt1.x, sbox->pt1.y,
            sbox->pt2.x - sbox->pt1.x, sbox->pt2.y - sbox->pt1.y);

      i = mfg_gaoi_setarea(dest_aoi, dbox->pt1.x, dbox->pt1.y,
            dbox->pt2.x - dbox->pt1.x, dbox->pt2.y - dbox->pt1.y);

      i += mfg_carea(src_aoi, dest_aoi);

      }



unsigned char * get_dm ( box_type * box  )
/*
 *  This function gets a dot matrix from the
 */
      {
      unsigned char * parray;
      int size;
      int numlines, numpixels;
      int n, i=0;
```

```
      size = ( box->pt2.x - box->pt1.x ) * ( box->pt2.y - box->pt1.y );
      parray = (unsigned char *) malloc ( size * sizeof(unsigned long) );

      if ( parray == NULL )
      {
            return NULL;
      }

      numlines = 1 + box->pt2.y - box->pt1.y;
      numpixels = 1 + box->pt2.x - box->pt1.x;


      i += mfg_gaoi_setarea(dm_aoi, box->pt1.x, box->pt1.y, numpixels,
numlines );

      for (n=0; n < numlines; n++)
            mfg_brhline (dm_aoi, 0, n, numpixels, &(parray[n*numpixels]));


      return ( parray );
      }



/*************************************************************************
 *
 *  put_dm() -- puts a dot matrix into the frame buffer of the
 *              Univision board
 *
 *************************************************************************/
void put_dm ( unsigned char * parray, box_type * box )

      {
      int numlines, numpixels;
      int n;

      numlines = 1 + box->pt2.y - box->pt1.y;
      numpixels = 1 + box->pt2.x - box->pt1.x;


      delay (2);

      n = mfg_gaoi_setarea(dm_aoi, box->pt1.x, box->pt1.y, numpixels, numlines
);


      for (n=0; n < numlines; n++)
            mfg_bwhline (dm_aoi, 0, n, numpixels, &(parray[n*numpixels]));

      }


void save_screen (void)
{

      char filename[14];
```

```
      if (get_string_from_user("Enter screen capture file name: ",filename) ==
0)
      {
            mfg_iorim(t_aoi,s_aoi, SIGNED);
            mfg_iorim(b_aoi,save_aoi, SIGNED);
            mfg_tiff_save(save_aoi , IFFCL_GRAY, filename, IFFCOMP_NONE);
      }
}
/*********************************************************************

  Name: MOUSE.C

  Description:

*********************************************************************/
#include <dos.h>
#include <stdlib.h>

#include "mouse.h"
#include "ntiga.h"


#define MMARGIN 10
#define SHIFT_CNT 3


/* Declare globals */
button_type  buttons;              /* left/right buttons */
point_type   mouse_xy;             /* mouse position */


/* Declare all local data -- file scope only */
static point_type   corner;                /* corner point for box drawing */
static box_type     track_box;       /* for dragging a box */
static box_type     drag_box_bias;  /* contains biases on cursor position */
static int  drawing_fixpt_box = 0;  /* box drawing flag */
static int  dragging_box=0;         /* box dragging flag */
static int  mouse_enabled = 1;      /* status of mouse cursor */
static union REGS       regs;              /* for interrupt calls */
static struct SREGS sregs;                 /* ditto */
static point_type   speed={1,1}; /* speed in mikeys */

/* Declare local functions */
void far _loadds _saveregs track_mouse ( void );
void redraw_fixpt_box ( void );
void redraw_drag_box ( void );


int initialize_mouse( void )
/*
 *    This function verifies presence of the mouse, and installs the event
 *    handler for mouse motion.  Note that it must be called after display
 *  initialization, so the size of the display is known
 */
      {
      void (far _loadds _saveregs * tmadd)() = track_mouse;

      /*
```

```
        *  send the reset command.  AX contains return.  if AX = 0,
        *  there is no mouse driver installed, so return an error.
        */
       regs.x.ax = RESET_MOUSE;
       int86(0x33, &regs, &regs);
       if ( regs.x.ax == 0 ) return 1;

       /* Now set horizontal range of motion */
       regs.x.cx = MMARGIN << SHIFT_CNT;
       regs.x.dx = ( display_size.x - MMARGIN ) << SHIFT_CNT;
       regs.x.ax = SET_HCLIP;          /* op code for horizontal range */
       int86 ( 0x33, &regs, &regs );   /* do it */

       /* Now set mouse vertical range of motion */
       regs.x.cx = MMARGIN << SHIFT_CNT;
       regs.x.dx = ( display_size.y - MMARGIN ) << SHIFT_CNT;
       regs.x.ax = SET_VCLIP;          /* op code for vertical range */
       int86 ( 0x33, &regs, &regs );   /* do it */

       /* And set tracking speed to default */
       change_mouse_speed ( 0, 0 );

       /* Find out where the mouse driver thinks it is */
       regs.x.ax = GET_STATUS;
       int86 ( 0x33, &regs, &regs );
       mouse_xy.x = regs.x.cx;
       mouse_xy.y = regs.x.dx;

       /*
        *    Now install the mouse tracking event handler.  It is called
        * track_mouse, at the end of this file.  The CX register contains
        * the enable mask.  All mouse events are enabled, including motion
        * and button presses/releases.  ES:DX points to the track_mouse
        * function.
        */
       regs.x.ax   = SET_HANDLER;   /* function code for install event handler
*/
       regs.x.cx   = 0x7F;          /* all events enabled */
       sregs.es    = FP_SEG ( tmadd );
       regs.x.dx   = FP_OFF ( tmadd );
       int86x ( 0x33, &regs, &regs, &sregs );

       /* Now tell display adaptor where to go and enable mouse */
       set_curs_shape ( 0 );
       set_cursattr ( fcolor, 0xFFFFFFFFL, 0x0020, 0x0020 ); /*  set cursor to
white */

       set_curs_xy ( mouse_xy.x, mouse_xy.y );
       set_curs_state ( 1 );

       mouse_enabled = 1;

       /*    All done */


       return 0;
       }
```

```
void kill_mouse( void )
/*  Just shoot it in the head */
      {
      regs.x.ax = RESET_MOUSE;
      int86(0x33, &regs, &regs);
      set_curs_state ( 0 );
      }


void enable_mouse ( void )
/*    Function enables cursor on the screen */
      {
      mouse_enabled = 1;
      set_mouse_position ( &mouse_xy );
      set_curs_state(1);
      }


void disable_mouse ( void )
/*  Disables cursor--necessary when doing anything in the RED plane */
      {
      mouse_enabled = 0;
      set_curs_state ( 0 );
      }


void change_mouse_speed ( int xinc, int yinc )
/*    This function allows the user to speed up/slow down the mouse */
      {

      /* change x ? */
      if ( xinc )
            {
            speed.x += xinc;
            if ( speed.x < 0 ) speed.x = 0;
            }

      /* change y ? */
      if ( yinc )
            {
            speed.y += yinc;
            if ( speed.y < 0 ) speed.y = 0;
            }

      /* Now set the mouse speed of motion */
      regs.x.cx = speed.x;
      regs.x.dx = speed.y;
    regs.x.ax = SET_SPEED;
      int86 ( 0x33, &regs, &regs );

      /* All done */
      return;
      }


void set_mouse_position ( point_type * point )
/*  This function will place the mouse at the point specified by point */
```

```
    {
    /* set mouse position indicators */
    mouse_xy.x = point->x;
    mouse_xy.y = point->y;

    /* Tell the mouse driver where to go */
    regs.x.ax = PUT_MOUSE;
    regs.x.cx = ( mouse_xy.x ) << SHIFT_CNT;
    regs.x.dx = ( mouse_xy.y ) << SHIFT_CNT;
    int86 ( 0x33, &regs, &regs );

    /* Now tell TIGA where to go */
    set_curs_xy ( mouse_xy.x, mouse_xy.y );

    /* Fini */
    }


/*************************************************************************
 *  These two functions, mouse_fixpt_box and redraw_fixpt_box, enable the
 *  user to set a corner point, then define a box by dragging the mouse.
 *************************************************************************/
box_type *  mouse_fixpt_box ( void )
/*    Allows defining a box by dragging the mouse */
    {
    unsigned int old_ppop;
    short tmp;

    /* Get current ppop, and set it to xor */
    /* By the way, ppop stands for pixel processing operation */
      old_ppop = get_ppop ();
      set_ppop ( 10 );                  /* op code for source xor dest */

    /* Now define the first box and draw it. */
    track_box.pt1.x = corner.x = mouse_xy.x;
    track_box.pt1.y = corner.y = mouse_xy.y;
    track_box.pt2.x = track_box.pt1.x;
    track_box.pt2.y = track_box.pt2.y;
    draw_box ( &track_box );

    /* Enable track_mouse box drawing and wait for left button release */
    drawing_fixpt_box = 1;
    while ( buttons.left );   /* Just let the event handler handle it */

    /* Disable box drawing and undraw leftover box */
    drawing_fixpt_box = 0;
    draw_box ( &track_box );

    /* The pt1 coordinate must be upper left corner.  Check x'es first */
    if ( track_box.pt1.x > track_box.pt2.x )
        {
        tmp = track_box.pt1.x;
        track_box.pt1.x = track_box.pt2.x;
        track_box.pt2.x = tmp;
        }

    /* Now check y's */
    if ( track_box.pt1.y > track_box.pt2.y )
```

```
        {
        tmp = track_box.pt1.y;
        track_box.pt1.y = track_box.pt2.y;
        track_box.pt2.y = tmp;
        }

    /* Fix ppop and return with a pointer to track_box */
      set_ppop ( old_ppop );
    return &track_box;
    }


void redraw_fixpt_box ( void )
/*
 *    Internal routine to destroy a box previously drawn, then redraws a new
 *  box based on current cursor position and previously defined fixed or
 *  corner point.
 */
      {
      /* destroy old box */
    draw_box ( &track_box );  /* xor ppop will undraw for us */

    /* Set up track_box x's */
    track_box.pt1.x = mouse_xy.x;
    track_box.pt2.x = mouse_xy.x;
    if ( mouse_xy.x > corner.x ) track_box.pt1.x = corner.x;
    else    track_box.pt2.x = corner.x;

    /* Do y's */
    track_box.pt1.y = mouse_xy.y;
    track_box.pt2.y = mouse_xy.y;
    if ( mouse_xy.y > corner.y )    track_box.pt1.y = corner.y;
    else                            track_box.pt2.y = corner.y;

      /* And draw the box */
    draw_box ( &track_box );
    return;
    }


/*************************************************************************
 *  These two functions, mouse_drag_box and redraw_drag_box, enable grabbing
 *  a box and moving it to another position on the screen.
 *************************************************************************/
point_type mouse_drag_box ( box_type * b )
/*
 *    This function drags a box around the screen while the
 *    left button is pressed.  The box passed in indicates the size
 *    of the box only.  This function will not return until the
 *    user releases the mouse button.  Return is the position of the
 *    mouse when the left button was released.
 */
      {
    short w, h;
    int old_ppop;

    /* Get current ppop, and set it to xor */
    /* By the way, ppop stands for pixel processing operation */
```

```
      old_ppop = get_ppop ();
      set_ppop ( 10 );                    /* op code for source xor dest */

    /* compute box width, height */
    w = b->pt2.x - b->pt1.x;
    h = b->pt2.y - b->pt1.y;

    /* Drag box will contain a set of biases for current mouse position */
    drag_box_bias.pt1.x = -w/2;
    drag_box_bias.pt1.y = -h/2;
    drag_box_bias.pt2.x =  w/2;
    drag_box_bias.pt2.y =  h/2;

    /* Now draw the box at current mouse position */
    track_box.pt1.x = mouse_xy.x + drag_box_bias.pt1.x;
    track_box.pt1.y = mouse_xy.y + drag_box_bias.pt1.y;
    track_box.pt2.x = mouse_xy.x + drag_box_bias.pt2.x;
    track_box.pt2.y = mouse_xy.y + drag_box_bias.pt2.y;
    draw_box ( &track_box );

    /* Now enable dragging and wait until left button is released */
    dragging_box = 1;
    while ( buttons.left );

    /* Disable box drawing, and undraw the box that was leftover */
    dragging_box = 0;
    draw_box ( &track_box );
      set_ppop ( old_ppop );      /* reset old ppop */

    /* Fini */
    return ( mouse_xy );
      }


void redraw_drag_box ( void )
/*
 *  This function is used during drag operations.  It undraws a previously
 *  drawn box, then redraws a box at the current cursor position.
 */
    {
    /* First, undraw the last box drawn */
    draw_box ( &track_box );    /* xor ppop will undraw it */

    /* Now set new box data */
    track_box.pt1.x = mouse_xy.x + drag_box_bias.pt1.x;
    track_box.pt1.y = mouse_xy.y + drag_box_bias.pt1.y;
    track_box.pt2.x = mouse_xy.x + drag_box_bias.pt2.x;
    track_box.pt2.y = mouse_xy.y + drag_box_bias.pt2.y;

    /* And draw the box */
    draw_box ( &track_box );
    }


int ismouse_in_box_array ( box_type boxes[], int num_boxes )
/*
 *  This function determines if the mouse is inside any of the boxes
 *  defined in the array boxes.  boxes has a total of num_boxes
```

```
 *  elements.  If the mouse is in one of the boxes, return the box
 *  index.  If not, return a negative number.
 */
      {
      int i;

      /* Loop thru each box in the array */
      for ( i=0; i<num_boxes; i++ )
            {

            /* and test to see if the mouse is in there */
        if ( ( mouse_xy.x >= boxes[i].pt1.x ) &&
             ( mouse_xy.y >= boxes[i].pt1.y ) &&
             ( mouse_xy.x <= boxes[i].pt2.x ) &&
             ( mouse_xy.y <= boxes[i].pt2.y ) )

                  /* if we get here, the mouse is in the current box */
                  return i;

            }     /* end loop thru boxes */

      /* If we get here, the mouse was not in any of the boxes. */
      return -1;
      }


/*  Mouse Event Handler */
void far _loadds _saveregs track_mouse ( void )
/*  This sucker acts like an interrupt service routine, but its' not.
 *    The mouse driver takes the interrupt, and can call an event handler.
 *    Mouse driver also sets up the registers with mouse state for the
 *    handler; that's actually quite nice.  The event handler must
 *  return with a far ret, not an IRET, so we can't use the interrupt
 *  keyword.  That's why there is so much gobbledygook in the definition.
 *  Anyway, this function serves as our event handler, and automatically
 *  moves the mouse cursor and keeps track of the buttons.  It is
 *  installed in initialize_mouse.
 */
      {
    int x, y, b;

    /*
     *  On invocation, 8086 registers contain the event mask and
     *  cursor position.  These assembly language instructions save
     *  the data for c access.  This is the only method found to
     *  work, but makes for non-portable code.  So be it.
     */
    _asm mov b, bx;
    _asm mov x, cx;
    _asm mov y, dx;

    /* Get the button status */
    buttons.left  = b & 0x01;
    buttons.right = b & 0x02;

    /* Don't update position if mouse is not enabled */
    if ( mouse_enabled )
        {
```

```
        /* set new x and y positions */
        mouse_xy.x = x >> SHIFT_CNT;
        mouse_xy.y = y >> SHIFT_CNT;

              /* Now move it on the screen */
              set_curs_xy ( mouse_xy.x, mouse_xy.y );

              /* If we're in box drawing mode, do it */
              if ( drawing_fixpt_box ) redraw_fixpt_box();
              if ( dragging_box ) redraw_drag_box();
            }   /* end mouse enabled test */

      /* All done */
      return;
      }

/*********************************************************************
 *
 *   NETSET.C
 *
 *    This file contains the entry point for the neural network
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "mouse.h"
#include "net.h"
#include "ntiga.h"
#include "vcrsmfg.h"


/* Declare globals */
int mag;                        /* magnification factor */
int  inmain;                              /* Indicates in main() function */
char * kbin_buf;                /* pointer to keyboard buffer */
WORD kbin;                                /* single character event buffer */
char ctbuf[120];
char filframe[16];                      /* frame file name */
char filfield[16];                      /* field file name */
char filexemp[16];            /* raw exemplar file name */
char filtrainset[16];          /* sized exemplar file name */
char filnet[16];                        /* Pointer to net data file name */
box_type display_box;         /* application uses this region */


/* Declare local functions */
static void exit_program ( void );
static void setup_filenames ( char * );


/* Local commands */
/*static command_type frame   =
      {   "FrameEdit",
              frame_editor,
            "Edit frame video characteristics"              };
*/
```

```
static command_type field   =
      {   "FieldEdit",
            field_editor,
            "Define video fields for net processing"  };
static command_type exemped =
      {     "ExEdit",
            exemplar_editor,
            "Identify and edit exemplar characters for training"  };
static command_type exempsz =
      {     "MakeTrain",
            exemplar_sizer,
            "Set standard size of exemplar characters and create training set"
      };
static command_type train   =
      {   "NetTrainer",
            train_net,
            "Train the neural network"                              };
static command_type eval    =
      {   "EvalNet",
            eval_net,
            "Evaluate neural network recognition performance"     };


/************************************************************************
 * This is the entry point for the program
 ************************************************************************
 */
int main( int argc, char * argv[] )
      {
      int done;

      /*
       * If argc > 1, the user has specified a file name prefix.
       * Otherwise, we will use TRIAL.xxx, where xxx is defined by the
       *    type of file (i.e. frm for frame, fld for field, ... )
       */
      if ( argc > 1 ) setup_filenames ( argv[1] );
      else setup_filenames ( "TRIAL" );

      /* Now run all initialization functions */
      if      ( initialize_mfg () ) printf ( "Error during MFG init\n" );
      else if ( initialize_tiga () ) printf ( "Error during TIGA init\n" );
      else if ( initialize_mouse() ) printf ( "Error during mouse init\n" );
      else if ( init_user () ) printf ( "Error during user interface init\n"
);
      else
            {
            /* register the exit routine */
            if ( atexit ( exit_program ) )
                  put_message_with_wait ( "Exit function was not registered\n"
);

            /* Now clear the display and initialize */
            clear_display ();
            mag = 2;

            /* Loop until user commands quit */
```

```
            do
                  {

                  /* Indicate to command processor that we're in main */
                  /* Causes exit command to be displayed */
                  inmain = 1;

                  /* The frame editor is always enabled */
/*                enable_command ( &frame ); */

                  /* If the previous steps' file exist, enable the command */
                  if ( fexist( filframe ))    enable_command (&field);
                  if ( fexist( filfield ))    enable_command (&exemped);
                  if ( fexist( filfield ))    enable_command (&exempsz);
                  if ( fexist( filtrainset )) enable_command (&train);
                  if ( fexist( filnet   ))       enable_command (&eval);

                  /* Now wait for the user to do something */
                  if ( wait_for_user_event() )
                        {
                        done = confirm ("Confirm Application Exit ") > 0;
                        clear_message ();
                        }
                  else done = 0;

                  } while ( NOT done );

            }

      exit_program();
      return 0;
      }


void exit_program ( void )
/*    This function will be called on any type of program termination */
      {

      /* Terminate tecon, tiga and mouse drivers */
      kill_mouse();
      kill_tiga();

      /* Finished with program */
      }



void setup_filenames ( char * f )
/*
 *  This function takes up to 8 characters pointed at by f, and builds
 *    all file names for the netsetup program
 */
      {
      int temp;

      /* Determine length of filename prefix.  If > 8 truncate to 8 */
      temp = strlen ( f );
      if ( temp > 8 )   * ( f + 9) = '\0';
```

```
        /* Now build the file names */
        strcpy ( filframe, f );              /* frame data file */
        strcat ( filframe, ".frm" );
        strcpy ( filfield, f );              /* field data file */
        strcat ( filfield, ".fld" );
        strcpy ( filexemp, f );              /* exemplar data file */
        strcat ( filexemp, ".exp" );
        strcpy ( filtrainset, f );              /* exemplar data file */
        strcat ( filtrainset, ".trn" );
        strcpy ( filnet, f );                      /* network data file */
        strcat ( filnet, ".net" );

        }
```

NNET.C

```
/* NEURAL NETWORK FUNCTION -- INPUTS NORMALIZED 0 -- 1      */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define  EXTERNAL extern
#include "net.h"
#include "network.h"

#define BETA 1.0f

float sigmoid ( float );


char recognize ( unsigned char * char_in, float * confidence )
        {
        float xmax = 0.0f, xnmax = 0.0f, xsum = 0.0f;
        int k, i, max = -1;
        unsigned char * ct=char_in, cmax=0;

        for ( i = 0; i < I; i++ )
                {
                if ( *ct > cmax ) cmax = *ct;
                ct++;
                }

        if ( cmax == 0 )          cmax = 1;

        ct = char_in;
        for ( i=0; i < I; i++ )
                input[i] = (float) (*ct++) / ( float ) cmax ;

        net();

        /* find the maximum value */
        for ( k=0; k < K; k++ )
                {
                xsum = xsum + output[k];
                if ( output[k] > xmax )
                        {
```

```
                max = k;
                xmax = output[k];
                }
        if ( ( output[k] < xmax ) && ( output[k] > xnmax ) )
                xnmax = output[k];
        }

    *confidence = xmax - xnmax;
    if ( max == -1 )
        return ('!');
    else
        return ( inchar [ max ] );
    }


void net ( void )
    {
    float x;
    int i, j, k, temp;

    for(j=0; j<J; ++j)
      {
      x = 0.0f;
      temp = j * I;
      for(i=0; i<I; ++i)
                x = x + w1[temp+i] * (input[i] - 0.5);
      hidden[j] = sigmoid(x-thetaj[j]);
      }

    for(k=0; k<K; ++k)
      {
      x = 0.0f;
      temp = k * J;
      for(j=0; j<J; ++j)
        x = x + w2[temp + j] * hidden[j];
      output[k] = sigmoid (x-thetak[k]);
      }
    }


/*  SIGMOID FUNCTION          */

float sigmoid ( float x )
    {
    float y;

    if((BETA * x) < -50.0f)
        y = 0.0f;
    else if((BETA * x) > 50.0f)
        y = 1.0f;
    else
        y = 1.0f / (1.0f + exp(-BETA * x));
    return (y);
    }


int make_net(void)
    {
```

```
      int error=0;

      error  = ( ( inchar = ( char * ) ( malloc(K)))  == NULL );
      error |= ( ( input  = (float * ) (malloc(I*(sizeof(float))))  == NULL
);
      error |= ( ( hidden = (float * ) (malloc(J*(sizeof(float))))  == NULL
);
      error |= ( ( output = (float * ) (malloc(K*(sizeof(float))))  == NULL
);
      error |= ( ( w1 = (float * ) (malloc(I*J*(sizeof(float))))== NULL );
      error |= ( ( w2 = (float * ) (malloc(J*K*(sizeof(float))))== NULL );
      error |= ( ( thetak = (float * ) (malloc(K*(sizeof(float))))  == NULL
);
      error |= ( ( thetaj = (float * ) (malloc(J*(sizeof(float))))  == NULL
);

      return ( error );
      }

void kill_net ( void )
      {
      free ( inchar ) ;
      free ( input );
      free ( hidden );
      free ( output );
      free ( w1 );
      free ( w2 );
      free ( thetak );
      free ( thetaj );
      }


int read_weights ( void )
      {
      int i, j, k;
      unsigned int temp;
      FILE *wts;
      char str[55];

      /* Try opening the file */
      if ( ( wts = fopen( filnet, "r") ) == NULL )
            return -1;

      /* Read in the net and character dimensions */
      if ( fscanf ( wts,"%d %d %d %d %d", &I, &J, &K, &trwidth, &trheight ) !=
5 )
            {
            put_message_with_wait ( "error on reading weights file\0" );
            fclose(wts);
            return(-1);
            }

      /* Now, get the memory for the net */
      if ( make_net() )
            {
            put_message_with_wait (" error during make_net\0");
            fclose(wts);
            return (-1);
```

```
        }

    /* Now, read each of the characters in the order they were trained on */
    for ( k=0; k<K; k++ )
        {
        if ( fscanf ( wts, "%x", &temp ) != 1 )
            {
            fclose(wts);
            put_message_with_wait ( "Error reading weights file
character\0" );
            return -1;
            }
        inchar [k] = (char) temp;
        }

    /* Read in the thetaj vector */
    for(j=0; j<J; j ++ )
        {
        if ( fscanf(wts, "%f", &(thetaj[j]) ) != 1 )
            {
            put_message_with_wait ( " error on thetaj read\0" );
            fclose(wts);
            return ( -1 );
            }
        }

    /* And the thetak vector */
    for(k=0; k<K; k++ )
        {
        if ( fscanf(wts, "%f", &(thetak[k]) ) != 1 )
            {
            fclose(wts);
            put_message_with_wait ( " error on thetak read\0" );
            return ( -1 );
            }
        }

    /* And the W1 array, i.e. INPUT * W1 - THETAJ = HIDDEN */
    for(j=0; j<J; j++ )
        {
        for(i=0; i<I; i++ )
            {
            if ( fscanf(wts, "%f", &(w1[j*I+i]) ) != 1 )
                {
                fclose(wts);
                sprintf(str, " error on w1 read %d\0",w1 );
                put_message_with_wait ( str );
                return ( -1 );
                }
            }
        }

    /* Finally, read in W2, i.e. HIDDEN * W2 - THETAK = OUTPUT */
    for(k=0; k<K; k++)
        {
        for(j=0; j<J; j++)
            {
            if ( fscanf(wts, "%f", &(w2[k*J+j]) ) != 1 )
```

```
                        {
                        fclose(wts);
                        put_message_with_wait ( " error on w2 read\0" );
                        return ( -1 );
                        }
                    }
                }

        /* All done */
        fclose(wts);
        return ( 0 );
        }
```

PREFILT.C

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>


/*****************************************************************
* Function that compares two integers.  Returns 1 if int1 > int2 *
*                                            -1 id int1 < int2 *
*                                             0 if int1 = int2 *
* Standard compare function for qsort call.  see qsort manual    *
*****************************************************************/
static int sintcmpr(i1,i2)
register unsigned char *i1,*i2;
{
    if (*i1 > *i2)
        return(1);
    else
        if (*i1 < *i2)
            return(-1);
        else
            return(0);
}



/*****************************************************************
* Returns median of list of numbers 9 numbers.                   *
*                                                                *
*                                                                *
*                                                                *
*****************************************************************/
static unsigned char Median(unsigned char *list)
{
    int sintcmpr();

    qsort((char *)list,9,sizeof(unsigned char),sintcmpr);
    return (list[4]);
}


/*****************************************************************
* 3x3 median filter used to remove noise. Takes pixel and 8      *
* neighbors and replaces pixel with median of 9 pixels           *
```

```
*                                                                    *
*                                                                    *
*********************************************************************/
static void NoiseReduce (unsigned char *image, unsigned int xdim, unsigned int
ydim)
{
    unsigned int  i, j;
       unsigned char r1[100], r2[100], r3[100], rt[100], *swap, a[9];
       unsigned char *row1 = r1;
       unsigned char *row2 = r2;
       unsigned char *row3 = r3;
       unsigned char *result = rt;

       memcpy (row1, image, xdim * sizeof (unsigned char));
       memcpy (row2, &image[xdim], xdim * sizeof (unsigned char));
       memcpy (row3, &image[xdim * 2], xdim * sizeof (unsigned char));
       for (j=1; j<ydim-1; j++)
       {
              result[0] = row2[0];
              result[xdim-1] = row2[xdim-1];
              for (i=1; i<xdim-1; i++)
              {
                        a[0] = row1[i-1];
                        a[1] = row1[i];
                        a[2] = row1[i+1];
                        a[3] = row2[i-1];
                        a[4] = row2[i];
                        a[5] = row2[i+1];
                        a[6] = row3[i-1];
                        a[7] = row3[i];
                        a[8] = row3[i+1];
                        result[i] = Median (a);
              }
              swap = row1;
              row1 = row2;
              row2 = row3;
              row3 = swap;
        memcpy (row3, &image[xdim * (j+2)], xdim * sizeof (unsigned char));
        memcpy (&image[xdim*(j)], result, xdim * sizeof (unsigned char));
    }

}
/*****************************************************************
* 3x3 edge detection filter used to enhance contrast by                *
* convolution with a balanced impuse response array.                   *
* The kernal is designed to strenghten weak vertical lines             *
* and enhance overall contrast.
                    *
* The following kernal is used:
                    *
*    2   1  0
                                        *
*    6       1 -6
                                        *
*    0  -1 -2
                                        *
*****************************************************************/
```

```
static void EdgeDetect (unsigned char *image, unsigned int xdim, unsigned int
ydim)
{
        unsigned int  i, j;
        int sum;
        unsigned char r1[100], r2[100], r3[100], rt[100], *swap;
        unsigned char *row1 = r1;
        unsigned char *row2 = r2;
        unsigned char *row3 = r3;
        unsigned char *result = rt;

        memcpy (row1, image, xdim * sizeof (unsigned char));
        memcpy (row2, &image[xdim], xdim * sizeof (unsigned char));
        memcpy (row3, &image[xdim * 2], xdim * sizeof (unsigned char));
        for (j=1; j<ydim-1; j++)
        {
                result[0] = row2[0];
                result[xdim-1] = row2[xdim-1];
                for (i=1; i<xdim-1; i++)
                {
                        sum = row1[i-1] * 2 + row1[i] *  1 + row1[i+1] *  0 +
                                row2[i-1] * 6 + row2[i] *  1 + row2[i+1] *
-6 +
                                row3[i-1] * 0 + row3[i] * -1 + row3[i+1] *
-2;

                        if (sum < 0)
                              result[i] = 0;
                        else
                        if (sum > 255)
                              result[i] = 254;/* limited to 254, was 255 HHH
11/23/94 */
                        else
                    result[i] = sum;

                }
                swap = row1;
                row1 = row2;
                row2 = row3;
                row3 = swap;
         memcpy (row3, &image[xdim * (j+2)], xdim * sizeof (unsigned char));
         memcpy (&image[xdim*(j)], result, xdim * sizeof (unsigned char));
    }

}




void pre_filter (unsigned char *image, unsigned int xdim, unsigned int ydim)
{
      NoiseReduce(image, xdim, ydim);
      EdgeDetect(image, xdim, ydim);
}
```

TECONRTH.C

```
/*************************************************************************

  Name: tecon.c

  Description:  This file contains initialization and exit code for the
                  Tecon Precision 20 AT card.

*************************************************************************/
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <direct.h>
#include <time.h>
#include <malloc.h>

#include "tecon.h"
#include "ntiga.h"


struct DEVICE device;

static void reset_tecon ( void );
static int get_lval ( float );


int initialize_tecon ( char * filname )
      {

      /* Reset the tecon precision 20 AT board */
      reset_tecon();

      /* Now load the device file */
      if ( load_device_data ( filname ) ) return -1;

      /* Now initialize the various tecon registers based on device data */
      set_hvsize        ( );
      set_holdoffs      ( );
      set_levels        ( );
      set_clock         ( );
      set_dclock        ( );

      /* Fini */
      return (0);
      }


void set_hvsize ( void )
      {
      outpw ( 0x200, 0xA000 );
      outpw ( 0x202, device.lines );
      outpw ( 0x200, 0x9000 );
      outpw ( 0x202, device.pixels );
      outpw ( 0x200, 0x8000 );
      }


void set_holdoffs ( void )
      {
```

```
        outpw ( 0x200, 0xC000 );
        outpw ( 0x202, device.holdoffs );
        outpw ( 0x200, 0x8000 );
        }


void set_levels ( void )
        {
        int lvhex;
        float vrt = device.white, vrb = device.black;

        lvhex = get_lval (vrt);
        outpw (HCONT_REG, VRTLOAD) ;          /* set vrt load bit */
        outpw (HCMD_REG, lvhex) ;             /* set vrt value */
        outpw (HCONT_REG, UNLOAD) ;          /* reset vrt load bit */

        lvhex = get_lval (vrb);
        outpw (HCONT_REG, VRBLOAD) ;          /* set vrb load bit */
        outpw (HCMD_REG, lvhex) ;             /* set vrb value */
        outpw (HCONT_REG, UNLOAD) ;           /* reset vrb load bit */
        }

/*********************************************************************
            get hex value from given voltage level
*********************************************************************/

int get_lval (float voltage)
  {
    float li, lv;
    int lvhex;

       li    = (LMAX - LMIN)/255.;
       lv = (LMAX - voltage)/li;

    lvhex = (int) (lv + 0.5);         /* convert to integer, round up */
    return (lvhex);
}



int set_clock ( void )
        {
        float  cmin, cmax, cv, ci, clkf = device.clock;
        int cvhex;

        /* check value of clkf (MODE0) */
        if (clkf >= CMIN0 && clkf <= CMAX0)
              {
              cmin = CMIN0; cmax = CMAX0;
              cvhex = 0x0000;                                    /* clear bit 8 for
/mode2 */
              }
        /* check value of clkf (MODE1) */
        else if (clkf >= CMIN1 && clkf <= CMAX1)
              {
              cmin = CMIN1; cmax = CMAX1;
              cvhex = 0x0100;                                    /* set bit 8 for
mode2 */
```

```
        }
    /* return error value out of range */
    else
        return (-1);

    /* calculate count increment in Khz */
    ci = (cmax-cmin)/255.;

    /* calculate value */
    cv = 255. - (clkf - cmin)/ci;

    /* round up */
    cvhex += (int) (cv + 0.5);

    outport (HCONT_REG, CLKLOAD);      /* set clock load bit */
    cvhex = outpw (HCMD_REG, cvhex);        /* set clock value */
    outpw (HCONT_REG, UNLOAD);       /* reset clock load bit */
    return 0;
    }


void set_dclock ( void )

    {
    float  cdv, cdi;
    int cdhex;

    /* calculate phase delay increment */
    cdi = CDMAX/255.;

    /* calculate value */
    cdv = device.dclk/cdi;

    /* round up and cast value */
    cdhex = (int) (cdv + .5);

    /* set phase delay load bit */
    outpw (HCONT_REG, CDLOAD);

    /* set delay value */
    outpw (HCMD_REG, cdhex);

    /* reset load bit */
    outpw (HCONT_REG, UNLOAD);
    }


int get_frame( box_type * box )
    {
    int i, j;
    time_t t, t1;
    WORD w, address;
    box_type b;
    BYTE * parray, btemp;
    int odd = 0;
    char buf[120];

    /* Command the grab */
```

```
        sprintf ( buf, "%s%4x", "device.command = ", device.command );
        put_message ( buf );
        outpw ( 0x200, device.command );
        t = time( & t );
        do {
              t1 = time ( & t1 );
              if ( ( t1 - t ) > 2 ) return -1;
              w = inpw ( 0x204 );
              } while ( ( w & 0x0C ) != 0x08 );

        /* Tell the board where we want to start reading */
        parray = (BYTE*) malloc ( device.pixels );
        if ( parray == NULL ) return -1;

        /* This set of loops puts the data to the Gxi card */
        address = device.pixels / 4 + 1;
        for ( j=0; j<device.lines; j++ )
              {
              outpw ( 0x202, address );
              for ( i=0; i<device.pixels; i++ )
                    {
                    if ( odd )
                          {
                          btemp = ( w & 0xFF00 ) >> 8;
                          if ( btemp < 4 ) btemp = 0;
                          parray[i] = btemp;
                          odd = 0;
                          }
                    else
                          {
                          w = inpw ( 0x200 );
                          btemp = w & 0xFF;
                          if ( btemp < 4 ) btemp = 0;
                          parray[i] = btemp;
                          odd = 1;
                          }
                    }
              for ( i=0; i<38; i++ )
                    sprintf ( &buf[3*i], "%2x ", parray[i] );
              put_message ( buf );
              wait_for_user_event();

              b.pt1.y = box -> pt1.y + j;
              b.pt2.y = box -> pt1.y + j;
              b.pt1.x = box -> pt1.x;
              b.pt2.x = box -> pt1.x + device.pixels;

              put_dm ( parray, &b );
              }
        free ( parray );
        return 0;
        }


void kill_tecon(void)
        {
        reset_tecon ();
```

```
}


int load_device_data ( char * filname )
      {
      FILE * f;

      f = fopen ( filname, "r" );
      if ( f == NULL )
            {
            /* The file does not exist.  Get device file from Tecon directory
*/
            printf ( "Using TECON DEVICE1.DEV\n" );
            f = fopen ( "C:\\TECON\\DRIVERS\\DEVICE1.DEV", "r" );
            if ( f == NULL ) return ( -1 );
            }

      fscanf(f,"%*[^|]%*c%[^\n]",device.name);
      fscanf(f,"%*[^|]%*c%[^\n]",device.comment);
      fscanf(f,"%*[^|]%*c%x",&device.command);
      fscanf(f,"%*[^|]%*c%d",&device.pixels);
      fscanf(f,"%*[^|]%*c%d",&device.lines);
      fscanf(f,"%*[^|]%*c%x",&device.holdoffs);
      fscanf(f,"%*[^|]%*c%f",&device.white);
      fscanf(f,"%*[^|]%*c%f",&device.black);
      fscanf(f,"%*[^|]%*c%f",&device.clock);
      fscanf(f,"%*[^|]%*c%d",&device.pedal);
      fscanf(f,"%*[^|]%*c%d",&device.videonum);
      fscanf(f,"%*[^|]%*c%f",&device.dclk);

      fclose ( f );
      return 0;
      }


void save_device_data ( char * filname )
      {
      FILE * f;

      f = fopen ( filname, "w" );
      if ( f == NULL )
            {
            printf ( "Error on opening %s for output.\n", filname );
            return;
            }

      /* Save current device to disk */
      fprintf(f,"DEVICE NAME |%s\n",device.name);
      fprintf(f,"    COMMENT |%s\n",device.comment);
      fprintf(f,"       TYPE |%x\n",device.command);
      fprintf(f,"     PIXELS |%d\n",device.pixels);
      fprintf(f,"      LINES |%d\n",device.lines);
      fprintf(f,"   HOLDOFFS |%x\n",device.holdoffs);
      fprintf(f,"WHITE LEVEL |%f\n",device.white);
      fprintf(f,"BLACK LEVEL |%f\n",device.black);
      fprintf(f,"SAMPLE RATE |%f\n",device.clock);
```

```
      fprintf(f,"  FOOTPEDAL |%d\n",device.pedal);
      fprintf(f,"VIDEO INPUT |%d\n",device.videonum);
      fprintf(f,"DELAY CLOCK |%f\n",device.dclk);

      fclose ( f );
      }


void reset_tecon ( void )
      {
      outpw ( 0x206, 0 );
   }
```

TIGA.C

```
#include <stdio.h>
#include <stdlib.h>
#include <process.h>



#include "mouse.h"
#include "ntiga.h"
#include "vcrsmfg.h"
#include "net.h"


CONFIG config;
FONTINFO font;
short oldmode;
point_type display_size;
unsigned long fcolor, bcolor;
PTR gsp_buf;
PALET p[256];




int initialize_tiga(void)
      {
      int n, itemp;


      /* Get old video mode so we can restore on exit */
      oldmode = get_videomode ( );



      /* Attempt to open the CD... */
      if (tiga_set(CD_OPEN)<0)
            {
            if ( ( itemp = spawnlp(P_WAIT,"tigacd.exe", "tigacd", "-i", NULL)
) < 0 )
                  {
                  printf ( "tigacd spawn failed: error code = %d\n", itemp);
                  return 1;
                  }
```

```
        else if ( tiga_set(CD_OPEN)<0 ) return (1);
        }


    /* initialize TIGA interface... */
    if ( set_videomode ( TIGA, INIT_GLOBALS | CLR_SCREEN ) < 0)
        {
        printf("FATAL ERROR - unable to set video mode.\n");
        kill_tiga ();
        return (1);
        }

    /* Load extended primitives... */
    if (install_primitives()<0)
        {
        printf("FATAL ERROR - can't initialize extended primitives\n");
        kill_tiga();
        return (1);
        }



    get_config ( &config );

    n = get_fontinfo ( 0, &font );


    display_size.x = config.mode.disp_hres;
    display_size.y = config.mode.disp_vres;


    fcolor = 0xFF;
    bcolor = 0x00;

    for (n=4; n < 256; n++)
        {
        p[n].r = p[n].g = p[n].b = p[n].i = (uchar) n;
        }

    /* Now set colors for palette entries 0, 1, 2 */
/*  p[0].r =   0; p[0].g = p[0].b = p[0].i = 0;
    p[1].g = 255; p[1].r = p[1].b = 255; p[1].i = 255;
    p[2].b = 255; p[2].r = p[2].g = 255; p[2].i = 0;

    set_palet(256, 0, p);
*/
    mfg_err_level(2);


    gsp_buf = gsp_malloc
        ( config.mode.disp_psize * config.mode.disp_hres / 8 );



    return 0;
    }
```

```
void kill_tiga( void )
      {
      clear_display();
      set_curs_state ( 0 );
      set_videomode (OFF_MODE, INIT);
      tiga_set ( CD_CLOSE );
      }


void clear_display ( void )
      {
      clear_screen ( bcolor );
      clear_framebuffer();;
      }


void out_text ( point_type p, char * c, int w, int reverse )
      {
      int i, padit = 0;
      char ctemp, buf[200];

      if ( w > 199 ) w = 199;
      for ( i=0; i<w; i++ )
            {
            ctemp = *c++;
            if ( ctemp == 0 || padit )
                  {
                  padit = 1;
                  buf[i]= 0x20;
                  }
            else buf[i] = ctemp;
            }
      buf[w] = '\0';

      if ( reverse )
            set_colors ( bcolor, fcolor );
      else
            set_colors ( fcolor, bcolor );
      i = text_out ( p.x, p.y, buf);


      }



void draw_box ( box_type * b )
      {
      short w, h, x, y;

      /* Set up */
      x = b->pt1.x;
      y = b->pt1.y;
      w = b->pt2.x - b->pt1.x;
      h = b->pt2.y - b->pt1.y;

      /* Limit w and h */
      if ( ( x + w ) > display_size.x )    w = display_size.x - b->pt1.x;
```

```
      if ( ( y + h ) > display_size.y )    h = display_size.y - b->pt1.y;

      /* Confirm OK to draw and do it */
      if ( ( w > 0 ) && ( h > 0 ) )
            {
            set_colors ( fcolor, bcolor );
            draw_rect ( w, h, x, y );

            }
      }


void draw_box_with_margin ( box_type * b, int margin )
/*    This function draws a box with a margin around box defined by b */
      {
      box_type bm;

      /* adjust coordinates of the box for the margin */
      bm.pt1.x = b->pt1.x - margin;
      bm.pt1.y = b->pt1.y - margin;
      bm.pt2.x = b->pt2.x + margin;
      bm.pt2.y = b->pt2.y + margin;

      /* and draw the box */
      draw_box ( &bm );


      }


void undraw_box_with_margin ( box_type * b, int margin )
/*  This function undraws a box with a margin around box defined by b */
      {
      box_type bm;

      /* adjust coordinates of the box for the margin */
      bm.pt1.x = b->pt1.x - margin;
      bm.pt1.y = b->pt1.y - margin;
      bm.pt2.x = b->pt2.x + margin;
      bm.pt2.y = b->pt2.y + margin;

      /* and draw the box */
      undraw_box ( &bm );
      }


void undraw_box ( box_type * b )
      {
      short w, h, x, y;

      /* Set up */
      x = b->pt1.x;
      y = b->pt1.y;
      w = b->pt2.x - b->pt1.x;
      h = b->pt2.y - b->pt1.y;

      /* Limit w and h */
      if ( ( x + w ) > display_size.x )
```

```
            w = display_size.x - b->pt1.x;
      if ( ( y + h ) > display_size.y )
            h = display_size.y - b->pt1.y;

      /* Confirm OK to draw and do it */
      if ( ( w > 0 ) && ( h > 0 ) )
            {
            set_colors ( bcolor, bcolor );
            draw_rect ( w, h, x, y );
            set_colors ( fcolor, bcolor );

            }
      }



void clear_box  ( box_type * b )
/*
 *  This function will overwrite a box with the background color
 *  in the currently selected plane
 */
      {
      short w, h, x, y;

      /* Set up */
      x = b->pt1.x;
      y = b->pt1.y;
      w = b->pt2.x - b->pt1.x + 1;
      h = b->pt2.y - b->pt1.y + 1;

      /* Limit w and h */
      if ( ( x + w ) > display_size.x )
            w = display_size.x - b->pt1.x;
      if ( ( y + h ) > display_size.y )
            h = display_size.y - b->pt1.y;

      /* Confirm OK to draw and do it */
      if ( ( w > 0 ) && ( h > 0 ) )
            {
            /* Blotto stuff in the overlay plane */
            set_colors ( bcolor, bcolor );
            fill_rect ( w, h, x, y );
            set_colors ( fcolor, bcolor );



            /* Now blotto stuff in the red plane */
            set_colors ( bcolor, bcolor );
            fill_rect ( w, h, x, y );

            /* Restore to previous state */
            set_colors ( fcolor, bcolor );

            clear_framebox (b);

            }

      }
```

```
int normalize_dm ( unsigned char * parray, int size )
      {
      float slope;
      unsigned char max = 0, min = 255;
      int i;

      for ( i=0; i<size; i++ )
            {
            if ( parray[i] > max ) max = parray[i];
            if ( parray[i] < min ) min = parray[i];
            }

      if ( max <= min )
            {
            printf("error on dot matrix normalization \n");
            return ( -1 );
            }

      slope = 255.0f / ((float)( max-min ));
      for ( i=0; i<size; i++ )
            parray[i] = ( unsigned char )( ( parray[i] - min ) * slope );

      return ( 0 );
      }


unsigned char * magnify_dm ( short mag, unsigned char * ch, short sizex, short
sizey )
      {
      unsigned char * cptr;
      unsigned char * chout;
      int size, x, y, xm, ym ;

      size = mag * sizex * mag * sizey;
      chout = (unsigned char * ) malloc ( size );
      if (chout == NULL)
            printf("Malloc Error: magnify_dm\n");
      cptr = chout;

      for ( y=0; y<sizey; y++ )
            for ( ym = 0; ym<mag; ym++ )
                  for ( x=0; x<sizex; x++ )
                        for ( xm=0; xm<mag; xm++ ) *cptr++ = ch[x+y*sizex];

      return ( chout );
      }
```

TRAIN.C

```
/*************************************************************************
 *  Neural Net Number Reader Training Algorithm                          *
 *                                                                       *
 *  Written by: Capt Edward Fix, 20 Jul 88                               *
 *              Armstrong Aerospace Medical Research Laboratory          *
 *                                                                       *
```

```
 *  Modified by:    J. Houchard, 23 Aug 91                                  *
 *                  SAIC                                                    *
 *                                                                          *
 ****************************************************************************
 *  This program creates and trains a neural network based on an
 *      exemplar set defined in a file which is read in.  Program output
 *      is written to another file, for subsequent reading for other
 *      processes.
 *
 *  Essentially, the program reads in the data, gets the network set up,
 *      then iterates through a back propagation function until the neural
 *      network output matches the exemplar set.  The final net coeffi-
 *      cients are written to a file.
 *
 *   Default filenames are:
 *       numbers.trn          training data set (for input)
 *       numbers.wts          trainer output weights data
 *   The user may specify a different file name on invocation; the
 *   default extensions will still be used though.  For instance,
 *   invoking the program with:
 *       VIDTRAIN CNUMBERS
 *   will look for CNUMBERS.TRN to get input, and write to weights
 *   data to CNUMBERS.WTS.  Don't specify an extension on the command
 *   line.
 ****************************************************************************/

#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <string.h>
#include <float.h>
#include <stdlib.h>
#include <time.h>


#define EXTERNAL extern

#include "ntiga.h"
#include "vcrsmfg.h"


#include "net.h"
#include "network.h"

#include "concave.h"



/* A little macro for the noise generator */
#define random2( min, max ) (min==max?0:((rand() % (int)((max) - (min))) +
(min)))

void pre_filter (unsigned char *, unsigned int , unsigned int );


/* Declare Globals */
int ntrain;                     /* # training char defined */
int ntrain_allocated;           /* # training chars allocated */
```

```
        }
    /* return error value out of range */
    else
        return (-1);

    /* calculate count increment in Khz */
    ci = (cmax-cmin)/255.;

    /* calculate value */
    cv = 255. - (clkf - cmin)/ci;

    /* round up */
    cvhex += (int) (cv + 0.5);

    outport (HCONT_REG, CLKLOAD);       /* set clock load bit */
    cvhex = outpw (HCMD_REG, cvhex);          /* set clock value */
    outpw (HCONT_REG, UNLOAD);        /* reset clock load bit */
    return 0;
    }


void set_dclock ( void )

    {
    float  cdv, cdi;
    int cdhex;

    /* calculate phase delay increment */
    cdi = CDMAX/255.;

    /* calculate value */
    cdv = device.dclk/cdi;

    /* round up and cast value */
    cdhex = (int) (cdv + .5);

    /* set phase delay load bit */
    outpw (HCONT_REG, CDLOAD);

    /* set delay value */
    outpw (HCMD_REG, cdhex);

    /* reset load bit */
    outpw (HCONT_REG, UNLOAD);
    }


int get_frame( box_type * box )
    {
    int i, j;
    time_t t, t1;
    WORD w, address;
    box_type b;
    BYTE * parray, btemp;
    int odd = 0;
    char buf[120];

    /* Command the grab */
```

```
        sprintf ( buf, "%s%4x", "device.command = ", device.command );
        put_message ( buf );
        outpw ( 0x200, device.command );
        t = time( & t );
        do {
              t1 = time ( & t1 );
              if ( ( t1 - t ) > 2 ) return -1;
              w = inpw ( 0x204 );
              } while ( ( w & 0x0C ) != 0x08 );

        /* Tell the board where we want to start reading */
        parray = (BYTE*) malloc ( device.pixels );
        if ( parray == NULL ) return -1;

        /* This set of loops puts the data to the Gxi card */
        address = device.pixels / 4 + 1;
        for ( j=0; j<device.lines; j++ )
              {
              outpw ( 0x202, address );
              for ( i=0; i<device.pixels; i++ )
                    {
                    if ( odd )
                          {
                          btemp = ( w & 0xFF00 ) >> 8;
                          if ( btemp < 4 ) btemp = 0;
                          parray[i] = btemp;
                          odd = 0;
                          }
                    else
                          {
                          w = inpw ( 0x200 );
                          btemp = w & 0xFF;
                          if ( btemp < 4 ) btemp = 0;
                          parray[i] = btemp;
                          odd = 1;
                          }
                    }
              for ( i=0; i<38; i++ )
                    sprintf ( &buf[3*i], "%2x ", parray[i] );
              put_message ( buf );
              wait_for_user_event();

              b.pt1.y = box -> pt1.y + j;
              b.pt2.y = box -> pt1.y + j;
              b.pt1.x = box -> pt1.x;
              b.pt2.x = box -> pt1.x + device.pixels;

              put_dm ( parray, &b );
              }
        free ( parray );
        return 0;
        }


void kill_tecon(void)
        {
        reset_tecon ();
```

```
}


int load_device_data ( char * filname )
      {
      FILE * f;

      f = fopen ( filname, "r" );
      if ( f == NULL )
            {
            /* The file does not exist.  Get device file from Tecon directory
*/
            printf ( "Using TECON DEVICE1.DEV\n" );
            f = fopen ( "C:\\TECON\\DRIVERS\\DEVICE1.DEV", "r" );
            if ( f == NULL ) return ( -1 );
            }

      fscanf(f,"%*[^|]%*c%[^\n]",device.name);
      fscanf(f,"%*[^|]%*c%[^\n]",device.comment);
      fscanf(f,"%*[^|]%*c%x",&device.command);
      fscanf(f,"%*[^|]%*c%d",&device.pixels);
      fscanf(f,"%*[^|]%*c%d",&device.lines);
      fscanf(f,"%*[^|]%*c%x",&device.holdoffs);
      fscanf(f,"%*[^|]%*c%f",&device.white);
      fscanf(f,"%*[^|]%*c%f",&device.black);
      fscanf(f,"%*[^|]%*c%f",&device.clock);
      fscanf(f,"%*[^|]%*c%d",&device.pedal);
      fscanf(f,"%*[^|]%*c%d",&device.videonum);
      fscanf(f,"%*[^|]%*c%f",&device.dclk);

      fclose ( f );
      return 0;
      }


void save_device_data ( char * filname )
      {
      FILE * f;

      f = fopen ( filname, "w" );
      if ( f == NULL )
            {
            printf ( "Error on opening %s for output.\n", filname );
            return;
            }

      /* Save current device to disk */
      fprintf(f,"DEVICE NAME |%s\n",device.name);
      fprintf(f,"    COMMENT |%s\n",device.comment);
      fprintf(f,"       TYPE |%x\n",device.command);
      fprintf(f,"     PIXELS |%d\n",device.pixels);
      fprintf(f,"      LINES |%d\n",device.lines);
      fprintf(f,"   HOLDOFFS |%x\n",device.holdoffs);
      fprintf(f,"WHITE LEVEL |%f\n",device.white);
      fprintf(f,"BLACK LEVEL |%f\n",device.black);
      fprintf(f,"SAMPLE RATE |%f\n",device.clock);
```

```
        fprintf(f,"  FOOTPEDAL |%d\n",device.pedal);
        fprintf(f,"VIDEO INPUT |%d\n",device.videonum);
        fprintf(f,"DELAY CLOCK |%f\n",device.dclk);

        fclose ( f );
        }


void reset_tecon ( void )
        {
        outpw ( 0x206, 0 );
    }
```

TIGA.C

```
#include <stdio.h>
#include <stdlib.h>
#include <process.h>



#include "mouse.h"
#include "ntiga.h"
#include "vcrsmfg.h"
#include "net.h"


CONFIG config;
FONTINFO font;
short oldmode;
point_type display_size;
unsigned long fcolor, bcolor;
PTR gsp_buf;
PALET p[256];




int initialize_tiga(void)
        {
        int n, itemp;


        /* Get old video mode so we can restore on exit */
        oldmode = get_videomode ( );



        /* Attempt to open the CD... */
        if (tiga_set(CD_OPEN)<0)
                {
                if ( ( itemp = spawnlp(P_WAIT,"tigacd.exe", "tigacd", "-i", NULL)
) < 0 )
                        {
                        printf ( "tigacd spawn failed: error code = %d\n", itemp);
                        return 1;
                        }
```

```
        else if ( tiga_set(CD_OPEN)<0 ) return (1);
        }


    /* initialize TIGA interface... */
    if ( set_videomode ( TIGA, INIT_GLOBALS | CLR_SCREEN ) < 0)
        {
        printf("FATAL ERROR - unable to set video mode.\n");
        kill_tiga ();
        return (1);
        }

    /* Load extended primitives... */
    if (install_primitives()<0)
        {
        printf("FATAL ERROR - can't initialize extended primitives\n");
        kill_tiga();
        return (1);
        }



    get_config ( &config );

    n = get_fontinfo ( 0, &font );


    display_size.x = config.mode.disp_hres;
    display_size.y = config.mode.disp_vres;


    fcolor = 0xFF;
    bcolor = 0x00;

    for (n=4; n < 256; n++)
        {
        p[n].r = p[n].g = p[n].b = p[n].i = (uchar) n;
        }

    /* Now set colors for palette entries 0, 1, 2 */
/*  p[0].r =   0; p[0].g = p[0].b = p[0].i = 0;
    p[1].g = 255; p[1].r = p[1].b = 255; p[1].i = 255;
    p[2].b = 255; p[2].r = p[2].g = 255; p[2].i = 0;

    set_palet(256, 0, p);
*/
    mfg_err_level(2);


    gsp_buf = gsp_malloc
        ( config.mode.disp_psize * config.mode.disp_hres / 8 );



    return 0;
    }
```

```
void kill_tiga( void )
      {
      clear_display();
      set_curs_state ( 0 );
      set_videomode (OFF_MODE, INIT);
      tiga_set ( CD_CLOSE );
      }


void clear_display ( void )
      {
      clear_screen ( bcolor );
      clear_framebuffer();;
      }


void out_text ( point_type p, char * c, int w, int reverse )
      {
      int i, padit = 0;
      char ctemp, buf[200];

      if ( w > 199 ) w = 199;
      for ( i=0; i<w; i++ )
            {
            ctemp = *c++;
            if ( ctemp == 0 || padit )
                  {
                  padit = 1;
                  buf[i]= 0x20;
                  }
            else buf[i] = ctemp;
            }
      buf[w] = '\0';

      if ( reverse )
            set_colors ( bcolor, fcolor );
      else
            set_colors ( fcolor, bcolor );
      i = text_out ( p.x, p.y, buf);


      }



void draw_box ( box_type * b )
      {
      short w, h, x, y;

      /* Set up */
      x = b->pt1.x;
      y = b->pt1.y;
      w = b->pt2.x - b->pt1.x;
      h = b->pt2.y - b->pt1.y;

      /* Limit w and h */
      if ( ( x + w ) > display_size.x )    w = display_size.x - b->pt1.x;
```

```
      if ( ( y + h ) > display_size.y ) h = display_size.y - b->pt1.y;

      /* Confirm OK to draw and do it */
      if ( ( w > 0 ) && ( h > 0 ) )
            {
            set_colors ( fcolor, bcolor );
            draw_rect ( w, h, x, y );

            }
      }


void draw_box_with_margin ( box_type * b, int margin )
/*    This function draws a box with a margin around box defined by b */
      {
      box_type bm;

      /* adjust coordinates of the box for the margin */
      bm.pt1.x = b->pt1.x - margin;
      bm.pt1.y = b->pt1.y - margin;
      bm.pt2.x = b->pt2.x + margin;
      bm.pt2.y = b->pt2.y + margin;

      /* and draw the box */
      draw_box ( &bm );


      }

void undraw_box_with_margin ( box_type * b, int margin )
/*  This function undraws a box with a margin around box defined by b */
      {
      box_type bm;

      /* adjust coordinates of the box for the margin */
      bm.pt1.x = b->pt1.x - margin;
      bm.pt1.y = b->pt1.y - margin;
      bm.pt2.x = b->pt2.x + margin;
      bm.pt2.y = b->pt2.y + margin;

      /* and draw the box */
      undraw_box ( &bm );
      }


void undraw_box ( box_type * b )
      {
      short w, h, x, y;

      /* Set up */
      x = b->pt1.x;
      y = b->pt1.y;
      w = b->pt2.x - b->pt1.x;
      h = b->pt2.y - b->pt1.y;

      /* Limit w and h */
      if ( ( x + w ) > display_size.x )
```

```
            w = display_size.x - b->pt1.x;
      if ( ( y + h ) > display_size.y )
            h = display_size.y - b->pt1.y;

      /* Confirm OK to draw and do it */
      if ( ( w > 0 ) && ( h > 0 ) )
            {
            set_colors ( bcolor, bcolor );
            draw_rect ( w, h, x, y );
            set_colors ( fcolor, bcolor );

            }
      }



void clear_box  ( box_type * b )
/*
 *  This function will overwrite a box with the background color
 *  in the currently selected plane
 */
      {
      short w, h, x, y;

      /* Set up */
      x = b->pt1.x;
      y = b->pt1.y;
      w = b->pt2.x - b->pt1.x + 1;
      h = b->pt2.y - b->pt1.y + 1;

      /* Limit w and h */
      if ( ( x + w ) > display_size.x )
            w = display_size.x - b->pt1.x;
      if ( ( y + h ) > display_size.y )
            h = display_size.y - b->pt1.y;

      /* Confirm OK to draw and do it */
      if ( ( w > 0 ) && ( h > 0 ) )
            {
            /* Blotto stuff in the overlay plane */
            set_colors ( bcolor, bcolor );
            fill_rect ( w, h, x, y );
            set_colors ( fcolor, bcolor );



            /* Now blotto stuff in the red plane */
            set_colors ( bcolor, bcolor );
            fill_rect ( w, h, x, y );

            /* Restore to previous state */
            set_colors ( fcolor, bcolor );

            clear_framebox (b);

            }

      }
```

```
int normalize_dm ( unsigned char * parray, int size )
      {
      float slope;
      unsigned char max = 0, min = 255;
      int i;

      for ( i=0; i<size; i++ )
            {
            if ( parray[i] > max ) max = parray[i];
            if ( parray[i] < min ) min = parray[i];
            }

      if ( max <= min )
            {
            printf("error on dot matrix normalization \n");
            return ( -1 );
            }

      slope = 255.0f / ((float)( max-min ));
      for ( i=0; i<size; i++ )
            parray[i] = ( unsigned char )( ( parray[i] - min ) * slope );

      return ( 0 );
      }


unsigned char * magnify_dm ( short mag, unsigned char * ch, short sizex, short
sizey )
      {
      unsigned char * cptr;
      unsigned char * chout;
      int size, x, y, xm, ym ;

      size = mag * sizex * mag * sizey;
      chout = (unsigned char * ) malloc ( size );
      if (chout == NULL)
            printf("Malloc Error: magnify_dm\n");
      cptr = chout;

      for ( y=0; y<sizey; y++ )
            for ( ym = 0; ym<mag; ym++ )
                  for ( x=0; x<sizex; x++ )
                        for ( xm=0; xm<mag; xm++ ) *cptr++ = ch[x+y*sizex];

      return ( chout );
      }
```

TRAIN.C

```
/*************************************************************************
 *  Neural Net Number Reader Training Algorithm                          *
 *                                                                       *
 *  Written by: Capt Edward Fix, 20 Jul 88                               *
 *              Armstrong Aerospace Medical Research Laboratory          *
 *                                                                       *
```

```
 *  Modified by:     J. Houchard, 23 Aug 91                              *
 *                   SAIC                                                *
 *                                                                       *
 *************************************************************************
 *  This program creates and trains a neural network based on an
 *      exemplar set defined in a file which is read in.  Program output
 *      is written to another file, for subsequent reading for other
 *      processes.
 *
 *  Essentially, the program reads in the data, gets the network set up,
 *      then iterates through a back propagation function until the neural
 *      network output matches the exemplar set.  The final net coeffi-
 *      cients are written to a file.
 *
 *   Default filenames are:
 *       numbers.trn          training data set (for input)
 *       numbers.wts          trainer output weights data
 *   The user may specify a different file name on invocation; the
 *   default extensions will still be used though.  For instance,
 *   invoking the program with:
 *       VIDTRAIN CNUMBERS
 *   will look for CNUMBERS.TRN to get input, and write to weights
 *   data to CNUMBERS.WTS.  Don't specify an extension on the command
 *   line.
 *************************************************************************/

#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <string.h>
#include <float.h>
#include <stdlib.h>
#include <time.h>


#define EXTERNAL extern

#include "ntiga.h"
#include "vcrsmfg.h"


#include "net.h"
#include "network.h"

#include "concave.h"



/* A little macro for the noise generator */
#define random2( min, max ) (min==max?0:((rand() % (int)((max) - (min))) +
(min)))

void pre_filter (unsigned char *, unsigned int , unsigned int );


/* Declare Globals */
int ntrain;                     /* # training char defined */
int ntrain_allocated;           /* # training chars allocated */
```

```
train_type * trainset;          /* pointer to training set array */

/* Declare locals to this file */
static short srcup = 0;
static box_type bsrc, bdest, bblob;
static point_type textpt;
static short x_inc, y_inc, x_base;
static int log_file_enable = 0;

/* These are for the noise corruption generator */
static int num_noise      = 6, noise_amplitude = 64;
static int num_background = 0, min_background  = 0,  max_background = 200;
static int num_foreground = 0, min_foreground  = 201, max_foreground = 255;
static int num_shift_pixels = 2;
static int blobs_enabled  = 1;
static int number_iterations = 9;
static int Max_Blob_Intensity = 240;

static int        init_trainer ( void );
static void       change_contrast ( BYTE, BYTE *, BYTE * );
static void       shift_dm_ud ( int, BYTE *, BYTE *, BYTE );
static void       shift_dm_rl ( int, BYTE *, BYTE *, BYTE );
static void       add_noise ( int, BYTE *, BYTE * );
static void       display_trainer_stuff ( int, BYTE *, BYTE *, char * );
static BYTE *     set_blob ( short, short, BYTE * );
static int        start_net_train ( void );
static int        set_noise_params ( void );
static int        set_blob_params ( void );
static int        set_foreground_params ( void );
static int        set_number_iterations ( void );
static int        set_background_params ( void );
static int        set_shift_params ( void );
static BYTE       Find_Average ( BYTE * );
static void       Shuffle_Trainset (void);

static BYTE       *dm_blob;
static short      pgon_color;

static command_type train_cmd =
      {     "TrainNet",
            start_net_train,
            "Start Training" };

static command_type noise_cmd =
      {     "SetNoise",
            set_noise_params,
            "Set Gaussian Noise Corruption Parameters" };

static command_type blob_cmd =
      {     "SetBlobs",
            set_blob_params,
            "Set Blob (Random Shape) Corruption Parameters" };

static command_type fore_cmd =
      {     "SetFore",
            set_foreground_params,
            "Set Foreground Intensity Parameters" };
```

```
static command_type iter_cmd =
      {     "SetIter",
            set_number_iterations,
            "Set number of iterations" };

static command_type back_cmd =
      {     "SetBack",
            set_background_params,
            "Set Background Intensity Parameters" };

static command_type shift_cmd =
      {     "SetShift",
            set_shift_params,
            "Set Character Shifting Parameters" };


int train_net ( void )
      {
      box_type tbox;

      tbox.pt1.x = 5;
      tbox.pt2.x = display_size.x - 5;
      tbox.pt1.y = 15;
      tbox.pt2.y = display_size.y - 125;

      clear_display ();
      put_screen_title ( "Net Trainer" );
      draw_box ( &tbox );


      /* Read in the network */
      if ( init_trainer () )
            {
            put_message_with_wait ( " Init trainer failed ! \0" );
            clear_message ( );
            return -1;
            }
      hold_weights();

      /* See if user wants to enable iteration logging */
      if ( confirm ( "Do you want to log iteration data ? " ) )
            log_file_enable = 1;
      else
            log_file_enable = 0;

      /* Command Processing Loop */
      do {
            enable_command ( &train_cmd );
            enable_command ( &iter_cmd );
            enable_command ( &noise_cmd );
            enable_command ( &blob_cmd );
/*          enable_command ( &fore_cmd );
            enable_command ( &back_cmd ); */
            enable_command ( &shift_cmd );
            } while ( NOT wait_for_user_event() );

      /* Shall we save */
      if (  ! fexist ( filnet ))
```

```
            save_weights ( );
      else if  ( confirm ( "Do you want to save net ?" ) > 0 )
            save_weights ( );

      /* And terminate ops.  These functions deallocate memory */
      kill_net ();
      kill_backprop();
      kill_train_set ();

      clear_message ();
      clear_display ( );
      /* Fini */
      return 0;
      }


int start_net_train ( void )
      {
      float accum_conf, worst_conf, best_conf = -1.0f, confidence;
      int count=0, lcount=0, i, k, stop = 0, num_iters, no_train = 1;
      int blob_intensity;
      int iteration;
      BYTE noise_level;
      BYTE ctemp, *dm_src, *dm_dst1, *dm_dst2;
      char tmp_char, pbuf[150];
      FILE * iter;

      /* Get weights from the hold buffer and init noise generator */
      get_weights();
      srand ( 1 );


      for ( k=0; k<ntrain; ++k )
            trainset[k].Average = Find_Average(trainset[k].dm_ptr);


      /* allocate for the temporary dot matrix buffer */
      dm_dst1 = ( BYTE * ) malloc ( I * sizeof(BYTE) );
      dm_dst2 = ( BYTE * ) malloc ( I * sizeof(BYTE) );
      dm_blob = ( BYTE * ) malloc ( I * sizeof(BYTE) );
      if ( ( dm_dst1 == NULL ) || ( dm_dst2 == NULL ) )
            {
            put_message_with_wait ( " Temp DM malloc failed ! \0" );
            clear_message ( );
            return -1;
            }

      /* Open the iteration data file */
      if ( log_file_enable )
            {
            iter = fopen ( "iter.dat", "w" );
            if ( iter == NULL )
                  {
                  log_file_enable = 0;
                  put_message_with_wait ( "Error opening iter.dat\0" );
                  clear_message ( );
                  }
            else
```

```
                {
                _strtime ( pbuf );
                fprintf ( iter, "\n\n    %s\n\n" , pbuf );
                }
          }


      num_iters =  number_iterations * ntrain;

      /* Here is the main training loop */
      do {
            /* Initialize stuff for this iteration */
            worst_conf = 1.0f;
            accum_conf = 0.0f;          /* init the cost data */
            _fpreset();
            srand ( rand() );


            noise_level = (BYTE) (noise_amplitude);


            for (iteration=0; iteration<=number_iterations; iteration++ )
            {

                  Shuffle_Trainset();

                  if (blobs_enabled)
                  {
                        blob_intensity = (BYTE) ( random2 ( 0,
Max_Blob_Intensity ) );
                        if ( blob_intensity < 4 ) blob_intensity = 0;
                        dm_blob = set_blob ( 0, blob_intensity, dm_blob );
                  }


                  /* This is the loop which trains on each training character
*/
                  for ( k=0; k<ntrain; ++k )
                  {
                        /* set up to point at the training character dot
matrix */
                        dm_src = trainset[k].dm_ptr;

                        for ( i=0; i<I; i++ )
                              dm_dst2[i] = dm_src[i];
                        pre_filter (dm_dst2, trwidth , trheight );


                        if (num_shift_pixels)
                        {
                              shift_dm_ud
                                    ( random2 ( -num_shift_pixels,
num_shift_pixels ),
                                      dm_dst2, dm_dst1, trainset[k].Average );
                              shift_dm_rl
                                    ( random2 ( -num_shift_pixels,
num_shift_pixels ),
                                      dm_dst1, dm_dst2, trainset[k].Average );
```

```
                        }
/*                      else
                              for ( i=0; i<I; i++ )
                                    dm_dst2[i] = dm_src[i];*/

                        if (blobs_enabled)
                              for ( i=0; i<I; i++ )
                                    if (dm_dst2[i] < dm_blob[i])
                                          dm_dst1[i] = dm_blob[i];
                                    else
                                          dm_dst1[i] = dm_dst2[i];
                        else
                              for ( i=0; i<I; i++ )
                                    dm_dst1[i] = dm_dst2[i];


                        dm_src = dm_dst1;


                        if (num_noise)
                              add_noise ( noise_level, dm_src, dm_dst2 );
                        else
                              for ( i=0; i<I; i++ )
                                    dm_dst2[i] = dm_src[i];


                        /* See what the net thinks it is and train */
                        tmp_char = recognize ( dm_dst2, &confidence );
                        if ( tmp_char != trainset[k].ch )
                              confidence = -confidence;
                        accum_conf += confidence;
                        if ( confidence < worst_conf )
                              worst_conf = confidence;
                        /*if ( ( best_conf > 0.5f ) && log_file_enable )
                        fprintf ( iter, "%s%c%s%c%s%3.3d%s%3.3d%s%3.3d\n",
                              "    bad char= ", trainset[k].ch,
                              ", nnreturn= ", tmp_char,
                              ", noise= ", noise_level,
                              ", fore= ", foreground,
                              ", back= ", background );*/

                        if ( !no_train ) backprop ( trainset[k].out_index );

                        /* Now display what happened */
                        if ( ( (count%3) == 0 ) && no_train )
                              display_trainer_stuff ( k, trainset[k].dm_ptr,
dm_dst2, &tmp_char );

                        /* We want to stop if the user hits escape */
                        if ( kbhit() )
                              {
                              ctemp = (char) getch();
                              if ( ctemp == '\x1B' ) stop = 1;
                              }
                        if ( stop ) break;
                  }   /* end trainset loop */
```

```
                if ( stop ) break;
            }  /* end noise_level loop */

            if ( stop ) break;
            accum_conf = accum_conf / num_iters;

            /*
             * If the current net has better average cost than the previously
             * saved net, we may want to save it.
             */
            if ( no_train )
                  {
                  no_train = 0;
                  if ( accum_conf > best_conf )
                        {
                        /* hold weights copies the net into a temporary buffer
*/
                        hold_weights();
                        best_conf = accum_conf;
                        lcount = 0;
                        }


                  /* output the string at the bottom of the page and file */
                  sprintf ( pbuf, "count=%5.5d, conf=%6.3f, worst conf=%6.3f,
best conf=%6.3f, bcount=%5.5d",
                        count, accum_conf, worst_conf, best_conf, lcount );
                  put_message ( pbuf );
                  if ( log_file_enable )
                        fprintf ( iter,
                              "count = %5.5d,  conf = %6.3f, best_conf =
%6.3f\n",
                              count, accum_conf, best_conf, lcount );
                  count++;
                  lcount++;
                  }
            else no_train = 1;

            } while ( (1 == 1) || ( accum_conf < 1.05f ) || ( worst_conf < 0.5
)
                        /* && ( count < 50 ) */ );

      /* Clean up */
      free ( dm_dst1 );
      free ( dm_dst2 );
      free ( dm_blob );

      if ( log_file_enable )
            {
            _strtime ( pbuf );
            fprintf ( iter, "\n\n   %s\n\n" , pbuf );
            fclose ( iter );
            }

/*    if ( ! stop ) put_message ("Trainer Done");
      else put_message ("Trainer Stopped");
*/
      return (0);
```

```
}


int init_trainer ( void )
      {
      /*  Create and initiailize the network and backprop  */


      if ( init_backprop() )
            {
            put_message_with_wait ( "Error returned from init_backprop\0" );
            return ( -1 );
            }

      /* set up the display stuff */

      /* this point is where the statistics will be printed */
      textpt.x = MARGIN;
      textpt.y = display_size.y - 50;

      x_inc = 60;
      y_inc = 30;
      x_base = 10 + 2 * MARGIN + trwidth;
      srcup = 0;

      return 0;
      }




BYTE        Find_Average ( BYTE * src)
{
      int y;
      unsigned long sum = 0;

      for ( y = 0; y < trheight * trwidth; y++ )
            sum = src[y] + sum;

      sum = sum / (trwidth * trheight);

      return (BYTE) sum;
}



void shift_dm_rl ( int nbits, BYTE * src, BYTE * dest, BYTE Average )
/*
 * This function corrupts a dot matrix (dm) by shifting columns
 * left or right.
 */
      {
      int x, y;        /* x is position in row, y is position in column */
      int ybias;

      if ( nbits >= 0 )    /* shift right */
```

```
            {
            for ( y = 0; y < trheight; y++ )
                  {
                  ybias = y * trwidth;
                  for ( x = (trwidth-1); x >=0; x-- )
                        {
                        if ( x < nbits ) dest [ x + ybias] = Average;
                        else dest [ x + ybias ] = src [ x - nbits + ybias ];
                        }
                  }
            }
      else if ( nbits < 0 )
            {
            for ( y = 0; y < trheight; y++ )
                  {
                  ybias = y * trwidth;
                  for ( x = 0; x < trwidth; x++ )
                        {
                        if ( x < trwidth + nbits )
                              dest [ x + ybias ] = src [ x - nbits + ybias ];
                        else dest [ x + ybias] = Average;
                        }
                  }
            }
      }


void shift_dm_ud ( int nbits, BYTE * src, BYTE * dest, BYTE Average )
       /*
      This function corrupts a dot matrix (dm) by shifting rows
       up or down by nbits.  nbits > 0 means shift down
       */
       {
       int x, y, ybias;   /* x is position in row, y is position in column */

       if ( nbits >= 0 )
              {
              for ( y = trheight-1; y >= 0; y-- )
                    {
                    ybias = y * trwidth;
                    for ( x = 0; x < trwidth; x++ )
                          {
                          if ( y < nbits )
                                dest [ x + ybias] = Average;
                          else
                                dest [ x + ybias ] = src [ (y-
nbits)*trwidth + x];
                          }
                    }
              }
       else if ( nbits < 0 )
              {
              for ( y = 0; y < trheight; y++ )
                    {
                    ybias = y * trwidth;
                    for ( x = 0; x < trwidth; x++ )
                          {
                          if ( y >= nbits + trheight )
```

```
                                        dest [ x + ybias] = Average;
                                else
                                        dest [ x + ybias ] = src [ (y-
nbits)*trwidth + x];
                        }
                }
        }
}


void add_noise ( int level, BYTE * src, BYTE * dest )
/*
 * This function adds noise to a dot matrix, converts and stores the
 * dot matrix in the networks input array.
 */
        {
        int i, b, itemp;

        for(i=0; i<I; ++i)
                {
                /* add the noise in, and convert to normalized float */
                if ( level == 0 ) dest[i] = src[i];
                else
                        {
                        b = ( (int) src[i] ) & 0xFF;
                        itemp = random2 ( -level, level ) + b;
                        if ( itemp < 4 ) itemp = 0;
                        else if ( itemp > 0xFF ) itemp = 0xFF;
                        dest[i] = (BYTE) itemp;
                        }
                }
        }


void change_contrast (  BYTE fore, BYTE * src, BYTE * dest )
/*
 * This function accepts a dot matrix pointed to by source, which
 * must be the high contrast character. For each pixel in src which
 * has the value 0xFF, the pixel is set to the value of fore in dest.
 */
        {
        int i;

        /* Now change the levels of foreground and background */
        for ( i = 0; i < I; i++ )
                {
                if ( *src++ == 0xFF )  *dest = fore;
                dest++;
                }
        }


static void swap_train (int exemp1, int exemp2)
{
        train_type temp;

        temp.ch = trainset[exemp1].ch;
```

```
        temp.out_index = trainset[exemp1].out_index;
        temp.Average = trainset[exemp1].Average;
        temp.dm_ptr = trainset[exemp1].dm_ptr;

        trainset[exemp1].ch = trainset[exemp2].ch;
        trainset[exemp1].out_index = trainset[exemp2].out_index;
        trainset[exemp1].Average = trainset[exemp2].Average;
        trainset[exemp1].dm_ptr = trainset[exemp2].dm_ptr;

        trainset[exemp2].ch = temp.ch;
        trainset[exemp2].out_index = temp.out_index;
        trainset[exemp2].Average = temp.Average;
        trainset[exemp2].dm_ptr = temp.dm_ptr;
}


/* function used to randomize exemplar set.  Randomly picks
      exemplar characters and swaps them.  This is done (2 * number
      of exemplar characters) times (this is an arbitrary number).
*/
static void       Shuffle_Trainset (void)
{
      int i;
      int exemp1, exemp2;

      for (i=0; i<2*ntrain; i++)
      {
            exemp1 = random2 (0, ntrain);
            exemp2 = random2 (0, ntrain);
            swap_train (exemp1, exemp2);
      }
}

void display_trainer_stuff ( int row, BYTE * src,
                                                      BYTE * dest, char * ch
)
      {
      short y;
      static point_type pcol, pchar;

      y = font.charhigh + 35 + MARGIN + (row+1) * y_inc;


      if ( !srcup )
            {
            bsrc.pt1.x = MARGIN;
            bsrc.pt2.x = bsrc.pt1.x + trwidth - 1;
            bsrc.pt1.y = y;
            bsrc.pt2.y = bsrc.pt1.y + trheight - 1;
            put_dm ( src, &bsrc );
            draw_box ( &bsrc );
            }

      bdest.pt1.x = x_base;
      bdest.pt2.x = bdest.pt1.x + trwidth - 1;
      bdest.pt1.y = y;
      bdest.pt2.y = y + trheight - 1;
      put_dm ( dest, &bdest );
```

```
        pchar.x = bdest.pt2.x + MARGIN/2;
        pchar.y = y;
        out_text ( pchar, ch, 1, NORMAL_TEXT );

        if ( row == ntrain-1 )
              {
              if ( srcup ) out_text ( pcol, " ", 1, NORMAL_TEXT );
              x_base += x_inc;
              if ( x_base + x_inc > display_size.x -10 )
                    x_base = 10 + 2 * MARGIN + trwidth;
              srcup = 1;
              pcol.y = 35 + MARGIN + y_inc;
              pcol.x = x_base + x_inc/2 - MARGIN/2 - font.charwide/2;;
              out_text ( pcol, "*", 1, NORMAL_TEXT );
              }

        }


void DrawProc(int y, int x1, int xn)
{

        int x;

        for (x=x1; x<=xn; x++)
        {
              dm_blob [y * trwidth + x] = pgon_color;
        }
}


BYTE * set_blob ( short back, short inten, BYTE *dm_blob )
        {
        Point2 b[12];
        int i, num_vert;

        bblob.pt1.x = x_base;
        bblob.pt1.y = MARGIN + 35;
        bblob.pt2.x = bblob.pt1.x + trwidth - 1;
        bblob.pt2.y = bblob.pt1.y + trheight - 1;

        Setup_World(0, 0, trwidth-1, trheight-1);

        /* Yes, w'ell due 3 to 12 sided blobs */
        num_vert = random2 ( 3, 12 );
        for ( i=0; i<num_vert; i++ )
              {
              /*b[i].x = random2 ( 0, trwidth-1 ) + bblob.pt1.x;
              b[i].y = random2 ( 0, trheight-1 ) + bblob.pt1.y; */
              b[i].x = random2 ( 0, trwidth-1 );
              b[i].y = random2 ( 0, trheight-1 );
              }

        /*set_colors ( back, back );
        fill_rect ( trwidth, trheight, bblob.pt1.x, bblob.pt1.y );
        delay ( 1 );
```

```
        set_colors ( inten, back );
        fill_polygon ( num_vert, b );
        delay ( 1 );
        set_colors ( fcolor, bcolor );
        return get_dm ( &bblob ); */

        memset (dm_blob, back, I * sizeof(BYTE));

        pgon_color = inten;
        Concave(num_vert, b, DrawProc);

        put_dm (dm_blob, &bblob);


        return dm_blob;
        }

int set_number_iterations ( void )
{
      char buf[100];
      int n;

      do {
            sprintf ( buf,
                  "Enter number of iterations ( 1 < x <= 200 ), default = %d:
",
                  number_iterations + 1);
            n = get_number_from_user ( buf );
            if ( n >= 0 ) number_iterations = n - 1;
            } while ( ( number_iterations <= 0 ) || ( number_iterations > 200
) );

      return 0;
}


int set_noise_params ( void )
      {
      char buf[100];
      int n;

      if ( ! confirm ( "Do you want noise corruption ? " ) )
            {
            num_noise = 0;
            return 0;
            }

/*    do {
            sprintf ( buf,
                  "Enter the number of noise iterations ( 1 < x <= 20 ),
default = %d: ",
                  num_noise+1 );
            n = get_number_from_user ( buf );
            if ( n >= 0 ) num_noise = n - 1;
            } while ( ( num_noise <= 0 ) || ( num_noise > 20 ) );
*/
      do    {
            sprintf ( buf,
```

```
                "Enter the max noise amplitude ( 0 < x < 240 ), default =
%d: ",
                noise_amplitude );
            n = get_number_from_user ( buf );
            if ( n >= 0 ) noise_amplitude = n;
            } while ( ( noise_amplitude ) <= 0 || ( noise_amplitude > 240 ) );

      return 0;
      }


int set_blob_params ( void )
      {

      char buf[100];
      int n;

      if ( confirm ( "Do you want blob corruption ? " ) )
            {
            blobs_enabled = 1;
            do {
                  sprintf ( buf,
                        "Enter maximum blob intensity ( 1 < x <= 255 ),
default = %d: ",
                        Max_Blob_Intensity );
                  n = get_number_from_user ( buf );
                  if ( n >= 0 ) Max_Blob_Intensity = n;
                  } while ( ( Max_Blob_Intensity <= 0 ) || (
Max_Blob_Intensity > 255 ) );
            return 0;
            }
      else blobs_enabled = 0;

      return 0;
      }


int set_foreground_params ( void )
      {
      char buf[100];
      int n;

      if ( ! confirm ( "Do you want varying foregrounds ? " ) )
            {
            num_foreground = 0;
            return 0;
            }

      do {
            sprintf ( buf,
                  "Enter the number of foreground iterations ( 1 < x <= 10 ),
default = %d: ",
                  num_foreground+1 );
            n = get_number_from_user ( buf );
            if ( n >= 0 ) num_foreground = n - 1;
            } while ( ( num_foreground <= 0 ) || ( num_foreground > 10 ) );

      do {
```

```
            sprintf ( buf,
                  "Enter the min foreground intensity ( 0 <= x <= 255 ),
default = %d: ",
                  min_foreground );
            n = get_number_from_user ( buf );
            if ( n >= 0 ) min_foreground = n;
            } while ( ( min_foreground < 0 ) || ( min_foreground > 255 ) );

      do {
            sprintf ( buf,
                  "Enter the max foreground intensity ( 0 <= x <= 255 ),
default = %d: ",
                  max_foreground );
            n = get_number_from_user ( buf );
            if ( n >= 0 ) max_foreground = n;
            } while ( ( max_foreground < 0 ) || ( max_foreground > 255 ) );

      return 0;
      }


int set_background_params ( void )
      {
      char buf[100];
      int n;

      if ( ! confirm ( "Do you want varying backgrounds ? " ) )
            {
            num_background = 0;
            return 0;
            }

      do {
            sprintf ( buf,
                  "Enter the number of background iterations ( 1 < x <= 10 ),
default = %d: ",
                  num_background + 1 );
            n = get_number_from_user ( buf );
            if ( n >= 0 ) num_background = n-1;
            } while ( ( num_background <= 0 ) || ( num_background > 10 ) );

      do {
            sprintf ( buf,
                  "Enter the min background intensity ( 0 <= x <= 255 ),
default = %d: ",
                  min_background );
            n = get_number_from_user ( buf );
            if ( n >= 0 ) min_background = n;
            } while ( ( min_background < 0 ) || ( min_background > 255 ) );

      do {
            sprintf ( buf,
                  "Enter the max background intensity ( 0 <= x <= 255 ),
default = %d: ",
                  max_background );
            n = get_number_from_user ( buf );
            if ( n >= 0 ) max_background = n;
            } while ( ( max_background < 0 ) || ( max_background > 255 ) );
```

```
      return 0;
      }


int set_shift_params ( void )
      {
      char buf[100];
      int n;

      if ( ! confirm ( "Do you want to enable character shifting backgrounds ?
" ) )
            {
            num_shift_pixels = 0;
            return 0;
            }

      do {
            sprintf ( buf,
                  "Enter the number of pixels to shift ( 1 < x <= 10 ),
default = %d: ",
                  num_shift_pixels );
            n = get_number_from_user ( buf );
            if ( n >= 0 ) num_shift_pixels = n;
            } while ( ( num_shift_pixels <= 0 ) || ( num_shift_pixels > 10 )
);

      return 0;
      }

/*****************************************************************************
 *
 *    USER.C
 *
 *    This file contains the bulk of routines for interfacing with the user,
 *    providing core functions for presenting messages and command options.
 *    It also provides the routines for keyboard and mouse initiated inputs
 *    and commands.
 *****************************************************************************
 */

#include <conio.h>
#include <ctype.h>
#include <dos.h>
#include <malloc.h>
#include <string.h>
#include <stdlib.h>

#include "mouse.h"
#include "net.h"
#include "ntiga.h"
#include "vcrsmfg.h"

#define MAX_HELP  10

char msg_buf[200] = { "MESSAGE:  " };
int  msg_buf_len;
```

```
static char keyboard_buf[20];           /* buffered kybd input goes here */
static point_type msg_start_point;      /* messages to user start here */
static int max_msg_len;                 /* max # of message characters */
static int ncmd_boxes;                  /* # of command boxes on screen */
static int max_cmds;                    /* max commands available */
static box_type * cmdboxes;             /* pointer to cmd box array */
static command_type ** commands;        /* pointer to active command array */
static int ncmds;                               /* current number of commands
*/
static box_type arrows[4];              /* space for arrow boxes */
static int arrow_boxes_ok=0;            /* flag indicating arrows are drawn */
static box_type help_boxes[MAX_HELP];     /* regions for which help can be
defined */
static int nhlpboxes;                           /* current number of help
boxes */
static char * helpmessages[MAX_HELP];     /* pointers to the messages */
static char yn[] = { " (y/n) " };       /* For confirmation messages */


int init_user ( void )
      {
      int i, x, y;

      /* set up the pointer to the global keyboard buffer */
      kbin_buf = keyboard_buf;

      /* Set up the message location at the bottom of the display */
      msg_start_point.x = MARGIN;
    msg_start_point.y = display_size.y - MARGIN - font.charhigh;
      max_msg_len = ( display_size.x - MARGIN - msg_start_point.x )
                        / font.charwide;

      /* Compute the maximum number of command boxes and allocate cmdboxes */
      ncmd_boxes = ( display_size.x - 2 * MARGIN ) / ( MARGIN +
                        ( MAX_CHARS_PER_CMD * font.charwide + 2*FONT_SPACING
) );
      cmdboxes = ( box_type * ) malloc ( ncmd_boxes * sizeof ( box_type ) );
      if ( cmdboxes == NULL )
            {
            printf ( "Error allocating %d cmdboxes\n", ncmd_boxes );
            return -1;
            }

      /* Now set up the command box array, just above the message */
      x = MARGIN;
      y = msg_start_point.y - MARGIN - font.charhigh - 2*FONT_SPACING;
      for ( i=0; i<ncmd_boxes; i++ )
            {
            cmdboxes[i].pt1.x = x;
            cmdboxes[i].pt1.y = y;
            cmdboxes[i].pt2.x = x + MAX_CHARS_PER_CMD * font.charwide
                                                + 2*FONT_SPACING;
            cmdboxes[i].pt2.y = y + font.charhigh + 2*FONT_SPACING;
            x += MAX_CHARS_PER_CMD * font.charwide + 2*FONT_SPACING + MARGIN;
            }
```

```
      /* Now set up the command array */
      max_cmds = ncmd_boxes;
      commands = ( command_type ** )
                        malloc ( max_cmds * sizeof ( command_type * ) );
      if ( commands == NULL )
            {
            printf ( "Error allocating command array with %d elements\n",
max_cmds );
            return -1;
            }

      /*
       *  Now set up the display box, defining the portion of the display the
       *  application can actually use
       */
      display_box.pt1.x = MARGIN;
      display_box.pt1.y = 2 * MARGIN + font.charhigh; /* space for screen
title */
      display_box.pt2.x = display_size.x - MARGIN;
      display_box.pt2.y = cmdboxes[0].pt1.y - MARGIN;

      /* Set up the box where the frame will go */
      frame_box.pt1.x = display_box.pt1.x;
      frame_box.pt1.y = display_box.pt1.y + font.charhigh;
      frame_box.pt2.x = frame_box.pt1.x + device.pixels - 1;
      frame_box.pt2.y = frame_box.pt1.y + device.lines - 1;

      /* Clear the help msg counter */
      nhlpboxes = 0;

      /* Fini */
      return 0;
      }



int get_number_from_user ( char * str )
/*
 *    This function gets a positive number from the user.  If the user
 *    hits escape, or user types no characters, return is negative.
 */
      {
      int n;

      /* Loop until user does the right thing */
      do    {
            /* Present the message */
            put_message ( str );

            /* Now wait for user to input something */
            if ( ( wait_for_buffered_kbin ( 5 ) ) <= 0 )
                  /* User hit escape or exit command */
                  return -1;

            /* Convert the string */
            n = atoi ( kbin_buf );
            if ( n < 0 )
                  /* Decoded to negative */
```

```
                put_message_with_wait ( "Please try again" );

            /* Continue looping til user gets it right */
            } while ( n < 0 );

      /* All done */
      clear_message();
      return n;
      }


int get_string_from_user ( char * msg, char *str )
/*
 *    This function gets a positive number from the user.  If the user
 *    hits escape, or user types no characters, return is negative.
 */
      {
      int n = 0;

      /* Present the message */
      put_message ( msg );

      /* Now wait for user to input something */
      if ( ( wait_for_buffered_kbin ( 13 ) ) <= 0 )
                  /* User hit escape or exit command */
            return -1;

      strcpy(str, kbin_buf);

      /* All done */
      clear_message();
      return n;
      }



int wait_for_buffered_kbin ( int length )
/*
 *  This function collects characters from the keyboard into the global
 *  keyboard buffer.  It returns the number of characters, or -1 if an
 *  escape was hit.
 */
      {
      int kbin_buf_index = 0, kbin_buf_length;
      unsigned int ichar;

      if ( length < 20 ) kbin_buf_length = length + 1;
      else kbin_buf_length = 20;

      /* Now loop until we get an escape (error exit) or a carriage return */
      do  {
            /* wait til we get a keyboard hit */
            while ( ! kbhit () );

            /* get the character */
            ichar = getch() & 0x7F;
```

```
            /* Is it a backspace */
            if ( ichar == 0x08 )
                  {
                  kbin_buf_index--;
                  if ( kbin_buf_index < 0 ) kbin_buf_index = 0;
                  kbin_buf[kbin_buf_index] = '\0';
                  append_message ( kbin_buf );
                  }

            /* or is it a printable character */
            else if ( ( isprint ( ichar ) ) &&
                        ( kbin_buf_index < kbin_buf_length-1 ) )
                  {
                  kbin_buf[kbin_buf_index++] = (char) ichar;
                  kbin_buf[kbin_buf_index] = '\0';
                  append_message ( kbin_buf );
                  }

            /* or is it an escape key */
            else if ( ichar == 0x1B )
                  return -1;

            /* Continue looping until we get a carriage return */
            } while ( ichar != 0x0D );

      /*
       *  All done.  Null terminate, kill the current message and return the
       *  character count
       */
      kbin_buf[kbin_buf_index] = '\0';
      clear_message();
      return kbin_buf_index;
      }


int confirm ( char * str )
/*
 *    This function will present the string, and ask the user to confirm
 *  with a y/n.  return is positive if y received.  return is 0 if n
 *  received.  return is negative to abort.
 */
      {
    int goty = 0, gotn = 0;

      /* Concatenate the y/n string with the string provided by caller */
    strcpy ( ctbuf, str );
    strcat ( ctbuf, yn );

      /* Now loop until user hits y or n.  If user hits escape, return n */
      do
            {
            msg_buf_len = 10 + strlen ( ctbuf );
            strcpy ( &msg_buf[10], ctbuf );
            out_text ( msg_start_point, msg_buf, max_msg_len, NORMAL_TEXT );

            if ( wait_for_buffered_kbin ( 1 ) < 0 ) return -1;
            if ( ( kbin_buf[0] == 'y' ) || ( kbin_buf[0] == 'Y' ) )
```

```
            goty = 1;
        else if ( ( kbin_buf[0] == 'n' ) || ( kbin_buf[0] == 'N' ) )
            gotn = 1;
        } while ( ( NOT goty ) && ( NOT gotn ) );

    /* Clear existing message and return true if user hit a y */
  clear_message ( );
    return goty;
    }


void register_help_message ( box_type * b, char * msg )
/*    This function will register a message for a given region on the
 *    display.  If the right button is pressed in the region, the help
 *    message is displayed
 */
    {
    /* verify we have room to store the data */
    if ( nhlpboxes + 1 < MAX_HELP )
        {
        /* Store the box data */
        help_boxes[nhlpboxes].pt1.x = b->pt1.x;
        help_boxes[nhlpboxes].pt2.x = b->pt2.x;
        help_boxes[nhlpboxes].pt1.y = b->pt1.y;
        help_boxes[nhlpboxes].pt2.y = b->pt2.y;

        /* Now store the character pointer */
        helpmessages[nhlpboxes] = msg;

        /* Increment the current count and return */
        nhlpboxes ++;

        }

    else

        /* insufficient storage for anymore regions */
        put_message_with_wait ( "Too many help box requests" );

    /* All done */
    }


void process_help_request ( void )
/*
 *  This function is called if the user hits the right mouse button
 *  while waiting for a user event.  It checks to see where the mouse
 *  is, and presents a help message which has been pre-registered by
 *  the application.  If no help message is available for the region,
 *  an innocuous message is presented til the right button is released.
 */
  {
    int index;

  /* Check command box help */
  index = ismouse_in_box_array ( cmdboxes, ncmds );
  if ( index >= 0 )
```

```
        put_message ( commands[index]->helpmsg );

    else
        {
        /* Not in a command box.  Try the help box array */
        index = ismouse_in_box_array ( help_boxes, nhlpboxes );
        if ( index >= 0 ) put_message ( helpmessages[index] );
        else put_message ( "Help not available for this region" );
        }   /* End else */

    /* Message is presented.  Wait til right button is released */
    while ( buttons.right );
    clear_message();

       /* Fini */
    }


void clear_help_messages ( void )
/* This function is called to destroy all currently defined help messages */
      {
      nhlpboxes = 0;
      }



/* Just a stupid (but required) function to process the exit/back command */
int ret1 ( void )
      { return 1; }



/*********************************************************************
 *    The following two functions, enable_command and wait_for_user_cmd
 *  work in close coordination with one another.  They combine to
 *  a context sensitive set of command options, which the application
 *  defines.
 *********************************************************************
 */

static int ncmds_old = 0;

static command_type appexit = {     "Exit",
                                                ret1,
                                                "Exit neural network setup
application" };
static command_type back = {  "Back",
                                                ret1,
                                                "Go back to previous command
level" };

void enable_command ( command_type * cmd )
/*
 *    This function accumulates the commands prior to presentation to the
 *    user.
 */
    {
```

```
        /* Verify we have command boxes left */

        if ( ncmds < max_cmds )
                {
                if ( ( commands[ncmds] != cmd ) || ( ncmds >= ncmds_old ) )
                        {
                        commands[ncmds] = cmd;
                            center_cmd_in_box ( cmd->cmdstring, &cmdboxes[ncmds]
);
                        }
                ncmds++;
                }
      }


int wait_for_user_event ( void )
/*
 *    This function is the user cmd processor.  As the name implies, it
 *    will wait for a user event and selectively return based on the
 *    context of the user event.  A left mouse button in a command box
 *    will execute the command.  A right mouse button in a command box
 *    will present the help message for the command.  If anywhere else,
 *    left mouse will simply be passed on to the application, while a right
 *    mouse is synonomous with exit or back.
 *
 *    Return is nonzero for an escape or back command.  Return is zero for
 *    continue.
 */
      {
    int index, done=0, cmd=0, quit=0, i;

      /* Enable back/exit command */
      if ( inmain )     enable_command ( &appexit );
    else            enable_command ( &back );
    inmain = 0;

       /* Now blotto old commands left over from previous command display */
       if ( ncmds_old > ncmds )
                {
                for ( i=ncmds; i<ncmds_old; i++ )
                        clear_box ( &cmdboxes[i] );
                }

      /* Init kbin discrete character */
      kbin = 0;

      /* Wait for the user to do something */
      do  {
            /* Is the left button pressed */
              if ( buttons.left )
                        {
                        /* left button hit.  see if in a command box */
                        index = ismouse_in_box_array ( cmdboxes, ncmds );

                    /* if not in a cmd box, let application have it */
                        if ( index < 0 ) done = 1;  /* not in command box */
                        else cmd = 1;                   /* in command box */
```

```
                    }   /* End buttons.left */

            if ( buttons.right )  /* Check help request */
                    process_help_request();

          /* Check the keyboard */
          if ( kbhit() )
                {
                /* User has hit a key.  Get it. */
                    kbin = getch() & 0xFF;
                /* Put extended characters in the upper 8 bits */
                    if ( ( kbin == 0 ) || ( kbin == 0xE0 ) )
                          kbin = ( getch() & 0xFF ) << 8;

                    /* If it was an escape, indicate exit */
                    if ( kbin == 0x1B ) quit = -1;
                    if ( kbin == 'p' || kbin == 'P')
                          save_screen();
                    /* Otherwise, pass it on to the application */
                    else done = 1;
                }

          /* continue looping til we get something */
            } while ( NOT done && NOT quit && NOT cmd );


       /* Init for next pass */
       ncmds_old = ncmds;
      ncmds = 0;

       /* If a command was issued, return result of the command */
       if ( cmd )
              {
              while ( buttons.left );     /* wait til user releases button */
              return (( * ( commands[index]->func ) ) ());
              }   /* end if cmd */

       /* Otherwise, return exit indicator */
       else return quit;
      }


void center_cmd_in_box ( char * str, box_type * box )
/*
 *    This function does what the name says, it draws a box, then centers
 */
      {
      int length;
      point_type p;

      /* First, blotto the old box and redraw the frame */
      clear_box ( box );

      draw_box ( box );

      /* Now compute the start coordinates for the centered text */
```

```
      length = strlen ( str );
      p.x = ( box->pt1.x + box->pt2.x ) / 2 - font.charwide * length / 2;
      p.y = ( box->pt1.y + box->pt2.y ) / 2 - font.charhigh / 2;
      out_text ( p, str, length, REVERSE_TEXT );
      }


void put_message ( char * str )
      {
       msg_buf_len = 10 + strlen ( str );
       strcpy ( &msg_buf[10], str );
       out_text ( msg_start_point, msg_buf, max_msg_len, NORMAL_TEXT );
      }


void put_message_with_wait ( char * msg )
/*  Puts a warning/error message and forces acknowlegdement */
      {

      /* Put the message out, with the hit any key appended */
      put_message ( msg );
      append_message ( " (Hit any key) " );

       /* Now wait til a key is hit or a button is pressed */
      wait_for_user_event ();

      /* and we're done */
      }


void append_message ( char * str )
      {
    strcpy ( &msg_buf[msg_buf_len], str );
      out_text ( msg_start_point, msg_buf, max_msg_len, NORMAL_TEXT );
      }


void put_error_message ( void )
      {
      out_text (  msg_start_point,
                        "Invalid entry: Hit any key to continue or <ESC> to
exit mode",
                        max_msg_len, REVERSE_TEXT );
    }


void clear_message ( void )
      {
      put_message ( "" );
      }


void put_screen_title ( char * t )
/*    This function puts the title (t) at the top of the screen */
      {
      point_type p;
      int l;
```

```
      l = strlen ( t );
      p.x = ( display_size.x + 1 ) / 2 - l * font.charwide / 2;
      p.y = MARGIN;
      out_text ( p, t, l, REVERSE_TEXT );
      }


void put_box_title ( char * t, box_type * b )
/*  This function puts a title above a box and draws the box */
      {
      point_type p;    /* where to put the string */
      int l;           /* length of string */

      /* Get the string length */
      l = strlen ( t );

    /* Now set up the coordinates to put the title */
    p.x = ( b->pt1.x + b->pt2.x ) / 2 - l * font.charwide / 2;
    p.y = b->pt1.y - font.charhigh - 1;

    /* Now put the text out, and draw the display box */
    draw_box_with_margin ( b, 1 );
       out_text ( p, t, l, NORMAL_TEXT );

    /* Fini */
       }


void connect_lines_in_box ( point_type * bias,
            point_type * a, point_type * b, point_type * c )

    {
    draw_line ( bias->x + a->x, bias->y + a->y,
                  bias->x + b->x, bias->y + b->y );
    draw_line ( bias->x + b->x, bias->y + b->y,
                  bias->x + c->x, bias->y + c->y );
    }


void init_arrows_edit_box ( box_type * b )
/*
 *  This function puts up/down/right/left arrows in the box pointed to by
 *  b.  If b is to small, it returns -1.  Otherwise, the arrows are drawn
 *  and return is 0;
 */
      {
      int temp, absize, i;
      short min, mid, max;
      point_type ul, uc, ur, cl, cr, ll, lc, lr;

      /* Compute the size of the arrow boxes */
      temp = ( b->pt2.y - b->pt1.y - 4 * MARGIN ) / 3;
      absize = ( b->pt2.x - b->pt1.x - 4 * MARGIN ) / 3;
      if ( temp < absize ) absize = temp;
```

```
        /* Now set up the arrow boxes.  Do upper left x'es first. */
        arrows[0].pt1.x = b->pt1.x/2 + b->pt2.x/2 - absize/2;
        arrows[1].pt1.x = arrows[0].pt1.x + absize + MARGIN;
        arrows[2].pt1.x = arrows[0].pt1.x;
        arrows[3].pt1.x = arrows[0].pt1.x - absize - MARGIN;

        /* Now do upper left y's */
        arrows[0].pt1.y = b->pt1.y + MARGIN;
        arrows[1].pt1.y = arrows[0].pt1.y + absize + MARGIN;
        arrows[2].pt1.y = arrows[0].pt1.y + 2 * ( absize + MARGIN );
        arrows[3].pt1.y = arrows[0].pt1.y + absize + MARGIN;

        for ( i=0; i<4; i++ )
              {
              arrows[i].pt2.x = arrows[i].pt1.x + absize;
              arrows[i].pt2.y = arrows[i].pt1.y + absize;
              draw_box ( &arrows[i] );
              }

        /* Now setup to draw the arrows */
         min = absize >> 3;
        mid = absize >> 1;
         max= absize - ( absize >> 3 );
        ul.x = min; ul.y = min; uc.x = mid; uc.y = min; ur.x = max; ur.y = min;
        cl.x = min; cl.y = mid; cr.x = max; cr.y = mid;
        ll.x = min; ll.y = max; lc.x = mid; lc.y = max; lr.x = max; lr.y = max;

         /* Now draw the arrow lines */
        connect_lines_in_box ( &arrows[UP_ARROW].pt1, &ll, &uc, &lr );
        connect_lines_in_box ( &arrows[RIGHT_ARROW].pt1, &ul, &cr, &ll );
        connect_lines_in_box ( &arrows[DOWN_ARROW].pt1, &ul, &lc, &ur );
        connect_lines_in_box ( &arrows[LEFT_ARROW].pt1, &ur, &cl, &lr );

        /* finally, fix the arrows box to be square, draw and put the title */
        b->pt1.x = arrows[LEFT_ARROW].pt1.x - MARGIN;
        b->pt1.y = arrows[UP_ARROW].pt1.y - MARGIN;
        b->pt2.x = arrows[RIGHT_ARROW].pt2.x + MARGIN;
        b->pt2.y = arrows[DOWN_ARROW].pt2.y + MARGIN;
        put_box_title ( "EDIT ARROWS", b );

        /* Now, indicate arrows are enabled */
        arrow_boxes_ok = 1;

        /* Fini */
        return;
        }


void kill_arrow_boxes ( void )
        {
        arrow_boxes_ok = 0;
        }



int check_edit_arrows ( void )
/*
```

```
 *  This function determines if the user has hit an arrow key or
 *  hit mouse in an arrows box.
 *
 *  Return >= 0:     Identifies the specific arrow
 *  Return < 0:      No arrow.
 */
      {
      int index;

      /* See if we should check the mouse initiated arrows */
      if ( arrow_boxes_ok )
            {
            /* See if left button is pressed */
            if ( buttons.left )
                  {
                  /* See if mouse is in an arrows box */
                  index = ismouse_in_box_array ( arrows, 4 );
                  if ( index >= 0 )
                        {
                        delay ( 100 );
                        return index;
                        }
                  }   /* end buttons.left test */
            }   /* end arrow_boxes_ok test */

      /* Now check keyboard */
      else if ( kbin != 0 )
            {
            /* See if the keyboard character is an arrow */
            if       ( kbin == 0x4800 )      return UP_ARROW;
            else if  ( kbin == 0x4D00 )      return RIGHT_ARROW;
            else if  ( kbin == 0x5000 )      return DOWN_ARROW;
            else if  ( kbin == 0x4B00 )      return LEFT_ARROW;
            }   /* end kbin test */

      /* No arrows have been hit */
      return -1;
   }
```

INCLUDE FILES

CONCAVE.H

```
#include "ggems.h"

void Setup_World (int, int, int, int);

void Concave(int , Point2 *,  void (*spanproc)());
```

GGEMS.H

```
/*
 * GraphicsGems.h
 * Version 1.0 - Andrew Glassner
 * from "Graphics Gems", Academic Press, 1990
 */
```

```
#ifndef GG_H

#define GG_H 1

/**********************/
/* 2d geometry types */
/**********************/

typedef struct Point2Struct { /* 2d point */
      double x, y;
      } Point2;
typedef Point2 Vector2;

typedef struct IntPoint2Struct {    /* 2d integer point */
      int x, y;
      } IntPoint2;

typedef struct Matrix3Struct {      /* 3-by-3 matrix */
      double element[3][3];
      } Matrix3;

typedef struct Box2dStruct {        /* 2d box */
      Point2 min, max;
      } Box2;


/**********************/
/* 3d geometry types */
/**********************/

typedef struct Point3Struct { /* 3d point */
      double x, y, z;
      } Point3;
typedef Point3 Vector3;

typedef struct IntPoint3Struct {    /* 3d integer point */
      int x, y, z;
      } IntPoint3;


typedef struct Matrix4Struct {      /* 4-by-4 matrix */
      double element[4][4];
      } Matrix4;

typedef struct Box3dStruct {        /* 3d box */
      Point3 min, max;
      } Box3;



/************************/
/* one-argument macros */
/************************/

/* absolute value of a */
#define ABS(a)          (((a)<0) ? -(a) : (a))

/* round a to nearest integer towards 0 */
```

```
#define FLOOR(a)        ((a)>0 ? (int)(a) : -(int)(-a))

/* round a to nearest integer away from 0 */
#define CEILING(a) \
((a)==(int)(a) ? (a) : (a)>0 ? 1+(int)(a) : -(1+(int)(-a)))

/* round a to nearest int */
#define ROUND(a)  ((a)>0 ? (int)(a+0.5) : -(int)(0.5-a))

/* take sign of a, either -1, 0, or 1 */
#define ZSGN(a)         (((a)<0) ? -1 : (a)>0 ? 1 : 0)

/* take binary sign of a, either -1, or 1 if >= 0 */
#define SGN(a)          (((a)<0) ? -1 : 0)

/* shout if something that should be true isn't */
#define ASSERT(x) \
if (!(x)) fprintf(stderr," Assert failed: x\n");

/* square a */
#define SQR(a)          ((a)*(a))


/***********************/
/* two-argument macros */
/***********************/

/* find minimum of a and b */
#define MIN(a,b)  (((a)<(b))?(a):(b))

/* find maximum of a and b */
#define MAX(a,b)  (((a)>(b))?(a):(b))

/* swap a and b (see Gem by Wyvill) */
#define SWAP(a,b) { a^=b; b^=a; a^=b; }

/* linear interpolation from l (when a=0) to h (when a=1)*/
/* (equal to (a*h)+((1-a)*l) */
#define LERP(a,l,h)     ((l)+(((h)-(l))*(a)))

/* clamp the input to the specified range */
#define CLAMP(v,l,h)    ((v)<(l) ? (l) : (v) > (h) ? (h) : v)


/*****************************/
/* memory allocation macros */
/*****************************/

/* create a new instance of a structure (see Gem by Hultquist) */
#define NEWSTRUCT(x)    (struct x *)(malloc((unsigned)sizeof(struct x)))

/* create a new instance of a type */
#define NEWTYPE(x)      (x *)(malloc((unsigned)sizeof(x)))


/********************/
/* useful constants */
/********************/
```

```
#define PI        3.141592    /* the venerable pi */
#define PITIMES2  6.283185    /* 2 * pi */
#define PIOVER2         1.570796    /* pi / 2 */
#define E         2.718282    /* the venerable e */
#define SQRT2           1.414214    /* sqrt(2) */
#define SQRT3           1.732051    /* sqrt(3) */
#define GOLDEN          1.618034    /* the golden ratio */
#define DTOR            0.017453    /* convert degrees to radians */
#define RTOD            57.29578    /* convert radians to degrees */


/************/
/* booleans */
/************/

#define TRUE            1
#define FALSE           0
#define ON        1
#define OFF             0
typedef int boolean;                /* boolean data type */
typedef boolean flag;               /* flag data type */

extern double V2SquaredLength(), V2Length();
extern double V2Dot(), V2DistanceBetween2Points();
extern Vector2 *V2Negate(), *V2Normalize(), *V2Scale(), *V2Add(), *V2Sub();
extern Vector2 *V2Lerp(), *V2Combine(), *V2Mul(), *V2MakePerpendicular();
extern Vector2 *V2New(), *V2Duplicate();
extern Point2 *V2MulPointByMatrix();
extern Matrix3 *V2MatMul();

extern double V3SquaredLength(), V3Length();
extern double V3Dot(), V3DistanceBetween2Points();
extern Vector3 *V3Normalize(), *V3Scale(), *V3Add(), *V3Sub();
extern Vector3 *V3Lerp(), *V3Combine(), *V3Mul(), *V3Cross();
extern Vector3 *V3New(), *V3Duplicate();
extern Point3 *V3MulPointByMatrix();
extern Matrix4 *V3MatMul();

extern double RegulaFalsi(), NewtonRaphson(), findroot();

#endif
```

LASDISC.H

```
int SetDiskFrame (int, unsigned long);

void SendOnLine (int);

void TermDisk (int);

int InitDisk (int);
```

MOUSE.H

```
#ifndef MOUSE_HEADER

      #define MOUSE_HEADER
```

```
#include "ntypes.h"

#define RESET_MOUSE             0
#define GET_STATUS              0x03
#define PUT_MOUSE               0x04
#define SET_HCLIP               0x07
#define SET_VCLIP               0x08
#define SET_HANDLER             0x0C
#define SET_SPEED               0x0F

/* Define mouse globals */
extern point_type mouse_xy;
extern button_type buttons;

/* Now declare functions defined in mouse.c */
int  initialize_mouse ( void );
void kill_mouse ( void );
void enable_mouse ( void );
void disable_mouse ( void );
void change_mouse_speed ( int, int );
void set_mouse_position ( point_type * );
box_type * mouse_fixpt_box ( void );
point_type mouse_drag_box ( box_type * );
int  ismouse_in_box_array ( box_type *, int );

#endif
```

NET.H

```
/* Global defines */
#define MARGIN              12
#define FONT_SPACING        5
#define MAX_FIELDS          5
#define MAX_EXEMPLARS       20
#define FIELD_SPACING       5
#define DONE                0
#define NOT                 !
#define UP_ARROW            0
#define     RIGHT_ARROW         1
#define     DOWN_ARROW          2
#define     LEFT_ARROW              3


/* Include program types */
#include "ntypes.h"

/* Here are global variable definitions */
extern int nfields;                 /* # fields currently defined */
extern int nexemp;                /* # exemplar chars defined */
extern int ntrain;                /* # training chars defined */
extern int nfields_allocated;     /* # fields currently allocated */
extern int nexemp_allocated;      /* # exemplar chars allocated */
extern int ntrain_allocated;      /* # exemplar chars allocated */
/* Data structure pointers */
extern field_type * fieldset;     /* pointer to fields array */
extern exemp_type * exempset;     /* pointer to exemplar set array */
extern train_type * trainset;     /* pointer to training set array */
```

```
/* Misc global data */
extern int mag;                    /* magnification factor */
extern int  inmain;                              /* Indicates in main()
function */
extern char * kbin_buf;            /* pointer to keyboard buffer */
extern WORD kbin;                                /* single character event
buffer */
extern char ctbuf[];                             /* temp place to put strings
*/
/* Misc global boxes */
extern box_type display_box;       /* application uses this region */
extern box_type frame_box;            /* Video frames will go here */
extern box_type field_box;                  /* field data is presented here */
/* File Names */
extern char filframe[];                     /* Ptr to frame file name */
extern char filfield[];                     /* Ptr to field file name */
extern char filnet[];                       /* Ptr to net data file name */
extern char filexemp[];            /* Ptr to raw exemp file name */
extern char filtrainset[];            /* Ptr to sized exemp file name */

/* And Here are function prototypes which are shared across source files */

/*    FRAME.C */
int  frame_editor ( void );

/* FIELD.C */
int  field_editor ( void );
int allocate_field_arrays ( field_type * );
void free_field_arrays ( field_type * );

/* FBOXES.C */
void fix_field_boxes ( int, int );
void highlight_field ( int );
void highlight_field_char ( int, int );
void unhighlight_field ( void );
void unhighlight_field_char ( void );
int  ismouse_in_field_box ( void );
int  ismouse_in_fieldchar_box ( int );
void draw_field ( int );
void undraw_field ( int );
void change_field_char ( int, int );
int  isfieldchar_selected ( void );
void get_selected_fieldchar_box ( box_type * );
void draw_frame_field ( int, unsigned long, unsigned int );
void draw_frame_char ( int, int, unsigned long, unsigned int );
void undraw_frame_field ( int, unsigned long ,unsigned int );
void undraw_frame_char ( int, int, unsigned long, unsigned int );

/* EXEMPID.C */
int  exemplar_identify ( void );

/* EXEMPSZ.C */
int exemplar_sizer ( void );

/* EXEMPED.C */
int  exemplar_editor ( void );

/* TRAIN.C */
```

```
int  train_net ( void );

/* EVAL.C */
int  eval_net ( void );

/* USER.C */
int  init_user ( void );
int  wait_for_buffered_kbin ( int );
int  get_number_from_user ( char * );
int  confirm ( char * );
void clear_help_messages ( void );
void register_help_message ( box_type * , char * );
void enable_command ( command_type * );
int  wait_for_user_event ( void );
void center_cmd_in_box ( char *, box_type * );
void put_message ( char * );
void put_message_with_wait ( char * );
void append_message ( char * );
void put_error_message ( void );
void clear_message ( void );
void put_screen_title ( char * );
void put_box_title ( char * , box_type * );
void init_arrows_edit_box ( box_type * );
void kill_arrow_boxes ( void );
int  check_edit_arrows ( void );
int get_string_from_user ( char * , char * );

/* FILES.C */
int get_field_set ( void );
void kill_field_set ( void );
int save_field_set ( void );
int  get_exemplar_set ( char * );
void kill_exemplar_set ( void );
int save_exemplar_set ( char * );
int  get_train_set ( char * );
void kill_train_set ( void );
int save_train_set ( char * );
int  fexist ( char * );
int save_exemp_as_train ( char *filename );
void kill_train_set ( void );
```

NETWORK.H

```
#ifndef NETWORK_HEADER

      #define NETWORK_HEADER

      /* #define DEBUGF */


      /* added these preprocessor macros because I is defined in
            one of the ITI MFG include files.
            4/22/94 RTH

      */
      #ifdef I
             #undef I
```

```
    #endif

    extern  int J, I, K, trwidth, trheight;
    extern  char  * inchar;
    extern  float * input;
    extern  float * hidden;
    extern  float * output;
    extern  float * w1;
    extern  float * w2;
    extern  float * thetaj;
    extern  float * thetak;

    /* BACKPROP.C */
    int   init_backprop ( void );
    void  backprop ( int );
    void  kill_backprop ( void );
    float net_cost ( int );
    int   save_weights ( void );
    void  hold_weights ( void );
    void  get_weights ( void );

    /* NET.C */
    char recognize ( unsigned char *, float * );
    void net ( void );
    int  make_net ( void );
    void kill_net ( void );
    int  read_weights ( void );

#endif
```

NTIGA.H

```
#ifndef NTIGA_HEADER

    #define NTIGA_HEADER

    #define NORMAL_TEXT 0
    #define REVERSE_TEXT 1


    #include <mfghost.h>

    #include "ntypes.h"

    extern CONFIG config;
    extern FONTINFO font;
    extern point_type display_size;
    extern unsigned long fcolor, bcolor;
    extern PTR gsp_buf;


    int initialize_tiga ( void );
    void kill_tiga ( void );
    void clear_display ( void );
    void out_text ( point_type , char * , int , int );
    void undraw_box ( box_type * ); /* box frame only */
    void draw_box ( box_type * );   /* box frame only */
```

```
      void draw_box_with_margin ( box_type * , int );
      void undraw_box_with_margin ( box_type * , int );
      void clear_box ( box_type * );  /* box fill / destroy */
      void copy_box ( box_type * , box_type * );  /* boxes must be same size
*/

#endif
```

NTYPES.H

```
#ifndef NTYPES_HEADER

      #define NTYPES_HEADER

      typedef unsigned char   BYTE;
      typedef unsigned int    WORD;
      typedef unsigned long   ULONG;

      #define MAX_CHARS_PER_CMD   10

      /*
       *    dm always refers to a dot matrix, i.e. an array which contains the
       *  grey scale values of each pixel.  ptr refers to a pointer.
       */

      typedef struct command_struct
            {
            char * cmdstring;       /* the string to present to the user */
            int ( * func )( void ); /* the function to call if commanded */
            char * helpmsg;         /* string to present for help */
            } command_type;

      typedef struct point_struct
            {
            short x;                                    /* x-coordinate of the
point */
            short y;                                    /* y-coordinate of the
point */
            } point_type;

      typedef struct box_struct
            {
            point_type pt1;              /* upper left  corner point */
            point_type pt2;              /* lower right corner point */
            }   box_type;

      typedef struct field_struct
            {
            short nchars;           /* number of characters in field */
            short cwidth;                       /* character width */
            short cheight;          /* character height */
            short drawn;                        /* indicates field has been
drawn */
            box_type fsbox;         /* box for displaying field in frame */
            box_type fdbox;         /* box for displaying field in field box
*/
            int nchars_allocated;   /* number of points in point array */
            point_type * points;    /* ptr to the array of start points */
```

```
            box_type * sbox;        /* ptr to array of src boxes in frame */
            box_type * dbox;        /* ptr to array of dest boxes on display
*/
            } field_type;

      typedef struct button_struct
            {
            short left;                   /* left  button on mouse */
            short right;                  /* right button on mouse */
            }  button_type;

      typedef struct train_struct
            {
            char  ch;             /* the character associated with this dm */
            int   out_index;        /* index into the output array */
            BYTE Average;                 /* average of pixels in dm */
            BYTE  * dm_ptr;         /* pointer to the dm */
            }   train_type;

      typedef struct exemp_struct
            {
            char ch;                /* the character associated with this dm
*/
            short cwidth;                       /* width of this character */
            short cheight;                      /* height of this character */
            short norm;                         /* normalized flag - set after
char edited */
            short drawn;                        /* set when boxes are valid
and drawn */
            BYTE * dm_ptr;          /* pointer to the dm */
            point_type ptchar;            /* point to place character at */
            box_type ebox;                      /* box bracketing entire
exemplar display */
            box_type sbox;                      /* screen box to put the dm */
            }   exemp_type;


#endif
```

STIMULUS.H

```
#define NOERROR          0
#define FILENOTFOUND    -1
#define EOFERROR        -2

unsigned int ShowNextStimulus(void);
int LoadStimulus(char * );
```

VCRSMFG.H

```
#include <stdio.h>

#ifndef VCRSMFG_HEADER
      #define VCRSMFG_HEADER

      #include "c:\tecon\inc\p20.h"
      #include "ntypes.h"
```

```
struct DEVICE {
      int num;                        /* Device number */
      char name[20];                  /* Device name */
      char comment[30];               /* Comment of device driver */
      unsigned int command;           /* Command to DVX-1024p */
      int pixels;                     /* X size of image */
      int lines;                      /* Y size of image */
      unsigned int holdoffs;          /* Holdoff value to DVX-1024p */
      float white;                    /* Upper fence sampling voltage */
      float black;                    /* Lower fence sampling voltage */
      float clock;                    /* Sampling rate (in Mhz) */
      int pedal;                      /* Pedal select */
      int videonum;                   /* Video input source to VMX-41 */
      float dclk;                     /* Clock start delay */
      };

extern struct DEVICE device;

int   initialize_mfg    ( void );
int   get_frame                ( void );
void clear_framebuffer (void);
void clear_framebox (box_type *box);


BYTE * get_dm ( box_type * );
void put_dm ( unsigned char * , box_type * );
void zoom_box ( box_type * , box_type *  ); /* copy pixels with mag */
void scale_box ( box_type * , box_type * );

#endif
```

Original Source Code Listing for Concave Polygon Scan Conversion

```
/*
Concave Polygon Scan Conversion
by Paul Heckbert
from "Graphics Gems", Academic Press, 1990
*/

/*
 * concave: scan convert nvert-sided concave non-simple polygon
 * with vertices at (point[i].x, point[i].y) for i in
 * [0..nvert-1] within the window win by
 * calling spanproc for each visible span of pixels.
 * Polygon can be clockwise or counterclockwise.
 * Algorithm does uniform point sampling at pixel centers.
 * Inside-outside test done by Jordan's rule: a point is
 * considered inside if an emanating ray intersects the polygon
 * an odd number of times.
 * drawproc should fill in pixels from xl to xr inclusive on scanline y,
 * e.g:
 * drawproc(y, xl, xr)
 * int y, xl, xr;
 * {
 *     int x;
 *     for (x=xl; x<=xr; x++)
 *             pixel_write(x, y, pixelvalue);
```

```
 * }
 *
 *  Paul Heckbert    30 June 81, 18 Dec 89
 */

#include <stdio.h>
#include <math.h>
#include "GraphicsGems.h"

#define ALLOC(ptr, type, n) \
          ASSERT(ptr = (type *)malloc((n)*sizeof(type)))

typedef struct {            /* window: a discrete 2-D rectangle */
    int x0, y0;                  /* xmin and ymin */
    int x1, y1;                  /* xmax and ymax (inclusive) */
} Window;


typedef struct {            /* a polygon edge */
    double x;
         /* x coordinate of edge's intersection with current scanline */
    double dx; /* change in x with respect to y */
    int i;     /* edge number: edge i goes from pt[i] to pt[i+1] */
} Edge;

static int n;               /* number of vertices */
static Point2 *pt;          /* vertices */

static int nact;            /* number of active edges */
static Edge *active; /* active edge list:edges crossing scanline y */

int compare_ind(), compare_active();

concave(nvert, point, win, spanproc)
int nvert;                  /* number of vertices */
Point2 *point;              /* vertices of polygon */
Window *win;                /* screen clipping window */
void (*spanproc)();         /* called for each span of pixels */
{
    int k, y0, y1, y, i, j, xl, xr;
    int *ind;  /* list of vertex indices, sorted by pt[ind[j]].y */

    n = nvert;
    pt = point;
    if (n<=0) return;
    ALLOC(ind, int, n);
    ALLOC(active, Edge, n);

    /* create y-sorted array of indices ind[k] into vertex list */
    for (k=0; k<n; k++)
         ind[k] = k;
    qsort(ind, n, sizeof ind[0], compare_ind);
                                        /* sort ind by pt[ind[k]].y */

    nact = 0;                       /* start with empty active list */
    k = 0;                          /* ind[k] is next vertex to process */
    y0 = MAX(win->y0, ceil(pt[ind[0]].y-.5));
```

```
                                                        /* ymin of
polygon */
    y1 = MIN(win->y1, floor(pt[ind[n-1]].y-.5));
                                                        /* ymax of
polygon */

    for (y=y0; y<=y1; y++) {  /* step through scanlines */

        /* scanline y is at y+.5 in continuous coordinates */
        /* Check vertices between previous scanline  */
        /* and current one, if any */

        for (; k<n && pt[ind[k]].y<=y+.5; k++) {
              /* to simplify, if pt.y=y+.5, pretend it's above */
              /* invariant: y-.5 < pt[i].y <= y+.5 */
        i = ind[k];
           /*
        * insert or delete edges before and after
              * vertex i  (i-1 to i, and i to i+1) from active
              * list if they cross scanline y
        */
        j = i>0 ? i-1 : n-1;     /* vertex previous to i */
        if (pt[j].y <= y-.5)
              /* old edge, remove from active list */
                    delete(j);
        else if (pt[j].y > y+.5)
              /* new edge, add to active list */
                    insert(j, y);
        j = i<n-1 ? i+1 : 0;     /* vertex next after i */
        if (pt[j].y <= y-.5)
              /* old edge, remove from active list */
                    delete(i);
        else if (pt[j].y > y+.5)
              /* new edge, add to active list */
                    insert(i, y);
        }

        /* sort active edge list by active[j].x */
        qsort(active, nact, sizeof active[0], compare_active);

        /* draw horizontal segments for scanline y */
        for (j=0; j<nact; j+=2) { /* draw horizontal segments */
        /* span 'tween j & j+1 is inside, span tween */
        /* j+1 & j+2 is outside */
        xl = ceil(active[j].x-.5);     /* left end of span */
        if (xl<win->x0) xl = win->x0;
        xr = floor(active[j+1].x-.5);
                                                        /* right end of
span */
        if (xr>win->x1) xr = win->x1;
        if (xl<=xr)
                        (*spanproc)(y, xl, xr);
                                                /* draw pixels in
span */
        active[j].x += active[j].dx;
                                                /* increment edge
coords */
        active[j+1].x += active[j+1].dx;
```

```
            }
        }
}


static delete(i)            /* remove edge i from active list */
int i;
{
    int j;

    for (j=0; j<nact && active[j].i!=i; j++);
    if (j>=nact) return;
         /* edge not in active list; happens at win->y0*/
    nact--;
    bcopy(&active[j+1], &active[j], (nact-j)*sizeof active[0]);
}

static insert(i, y)         /* append edge i to end of active list */
int i, y;
{
    int j;
    double dx;
    Point2 *p, *q;

    j = i<n-1 ? i+1 : 0;
    if (pt[i].y < pt[j].y) {p = &pt[i]; q = &pt[j];}
    else           {p = &pt[j]; q = &pt[i];}
    /* initialize x position at intersection of edge with scanline y */
    active[nact].dx = dx = (q->x-p->x)/(q->y-p->y);
    active[nact].x = dx*(y+.5-p->y)+p->x;
    active[nact].i = i;
    nact++;
}

/* comparison routines for qsort */
compare_ind(u, v) int *u, *v; {return pt[*u].y <= pt[*v].y ? -1 : 1;}
compare_active(u, v) Edge *u, *v; {return u->x <= v->x ? -1 : 1;}
```